AD-771 739

SAP: A MODEL FOR THE SYNTACTIC ANALYSIS OF PICTURES

Armond David Inselberg, et al

Washington University

Prepared for:

Advanced Research Projects Agency
Department of Defense
National Institutes of Health

June 1968

Unclassified
Security Classification

AD 771 239

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Computer Systems Laboratory, Washington University, St. Louis, Missouri	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

SAP: A Model for the Syntactic Analysis of Pictures

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Interim

5. AUTHOR(S) (First name, middle initial, last name)

Armond David Inselberg and Raymond M. Kline

6. REPORT DATE June, 1968	7a. TOTAL NO. OF PAGES ~~125~~ 133	7b. NO. OF REFS 56

8a. CONTRACT OR GRANT NO. (1) DOD(ARPA) Contract SD-302 (2) NIH(DRFR) Grant No. FR-00218	9a. ORIGINATOR'S REPORT NUMBER(S) Technical Report No. 9
b. PROJECT NO. (1) ARPA Project Code No. 5880	
c. Order No. 655	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY ARPA – Information Processing Techniques Washington, D.C. N.I.H., Div. of Research

13. ABSTRACT

A syntax-directed model is presented which is able to recognize and generate two-dimensional pictures while allowing a high degree of man/machine interaction. Starting with a field of points representing a picture, a string of symbols providing a structural description of the picture is produced by the syntactic component. The structural description, composed of higher level primitives (e.g., geometric symbols such as triangles and rectangles) and syntactic relations which exist between the primitives, is operated upon by the semantic component to provide a semantic interpretation for the picture. The syntactic component consists of a lexicon, a modified context-sensitive phrase structure grammar, and a set of transformation rules. The semantic component consists of a set of heuristics to abstract the picture and a modified context-sensitive phrase structure grammar which allows contextual restrictions to be applied to combinations of constituents existing at different levels of the syntax tree which syntactically describes the picture. Various aspects of the model have been programmed on the LINC (a small digital computer), the IBM 360/50, and the IBM 7072.

DD FORM 1 NOV 65 1473 REPLACES DD FORM 1473, 1 JAN 64, WHICH IS OBSOLETE FOR ARMY USE.

Security Classification

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Syntactic Analysis of Pictures						
SAP						
Scenic Input						
Syntax Directed Model						
Syntactic Functions						
Syntactic Component						
Higher Level Primitives						
Triplet Set						
Partial Ordering						
Parsing of Triplet Set						
Syntax Tree						
Semantic Component						
Semantic Interpretation						
Graphic Feedback						
Syntactic String						
Reverse Polish String						

Security Classification

SAP: A MODEL FOR THE SYNTACTIC ANALYSIS OF PICTURES

Armond David Inselberg and Raymond M. Kline

TECHNICAL REPORT NO. 9

June, 1968

Computer Systems Laboratory
Washington University
St. Louis, Missouri

This work has been supported by the Advanced Research Projects Agency of the Department of Defense under contract SD-302 and by the Division of Research Facilities and Resources of the National Institutes of Health under Grant FR-00218.

1b

"We are like dwarfs seated on the shoulders of giants; we see more things than the ancients and things more distant, but this is due neither to the sharpness of our own sight, nor to the greatness of our own stature, but because we are raised and borne aloft on that giant mass."

Bernard of Chartres

ABSTRACT

A syntax-directed model is presented which is able to recognize and generate two-dimensional pictures while allowing a high degree of man/machine interaction. Starting with a field of points representing the picture, a string of symbols providing a structural description of the picture is produced by the syntactic component. The structural description, composed of higher level primitives (e.g., geometric symbols such as triangles and rectangles) and syntactic relations which exist between the primitives, is operated upon by the semantic component to provide a semantic interpretation for the picture. The syntactic component consists of a lexicon, a modified context-sensitive phrase structure grammar, and a set of transformation rules. The semantic component consists of a set of heuristics to abstract the picture and a modified context-sensitive phrase structure grammar which allows contextual restrictions to be applied to combinations of constituents existing at different levels of the syntax tree which syntactically describes the picture. Various aspects of the model have been programmed on the LINC (a small digital computer), the IBM 360/50, and the IBM 7072.

LIST OF FIGURES

TABLE OF CONTENTS

SAP: A MODEL FOR THE SYNTACTIC ANALYSIS OF PICTURES

1. INTRODUCTION

The syntactic analysis of natural language has become a well-established technique in the field of linguistics. Whether a similar analysis can successfully be applied to pictures remains to be seen. This report is an effort to consider some of the problems which arise in the syntax-directed analysis of pictorial data. While it is seen that the syntactic-semantic approach may be used in both the analysis of linguistic and pictorial data, many of the formulations developed in linguistics should not be expected to carry over to pictures.

A most satisfactory situation would be one in which the computer has the ability to perform as well as a human in the field of pattern recognition. While there are industrial problems which require the machine to make discriminations beyond that which is humanly possible, a pattern recognizer which could recognize checkbook signatures, postage stamps, or airplane silhouettes would be no small achievement. Thus, the field of psychology may be an information source for pattern recognition in terms of such work as has been done by the Gestalt psychologists. However, a distinction must be maintained between a model which simulates human intelligence and a model which can provide the same results as human intelligence.

In developing this report, the pictorial data was restricted to straight line figures which have the semblance of cartoon-like drawings with no field of depth. Thus, the pictures may be considered to represent three-dimensional objects projected onto a plane with the point of view of the object being perpendicular to any one of its major axes. In the following chapters there is no distinction made between the terms *graphics* and *pictures* implying that the digitized points of the pictorial data can be the result of drawing on a cathode-ray tube with a light pen or scanning a hard copy photograph.

As a formal definition, the picture which is presented to the computer is called a *scene*. The scene, in turn, is composed of *figures*. The figures are built up from constituents called *primitives*. The higher level primitives for the examples used in this report are geometric symbols, such as triangles, circle, etc., and the figures are houses, trees, etc. The primitives are combined to form the figures by *syntactic relations*. The syntactic relations used in the present examples are *on top of*, etc.

This report presents a model which provides an approach to the syntactic analysis of pictures. As indicated earlier, the *pictures* are two-dimensional patterns which have significance in the *real-world*. The word *analyzer* is meant to indicate the ability to both recognize and generate pictures. *Syntactic* analysis indicates that a structural description is obtained, describing the topological features of the picture. It is on the basis of the structural description that the pattern recognition (called *semantic interpretation*) is accomplished.

The model, SAP (*S*yntactic *A*nalyzer of *P*ictures), was developed based on an interest in the general methodology and philosophy of syntax-directed analysis, and as such, provides an overall view of the problem. The model, as described in this report, is composed of two major components, a syntactic component and a semantic component. A syntax-directed meta-language to facilitate man/machine interaction is also described in detail.

An outline of SAP and the ability of the user to interact is represented in Figure 1. The set of two-dimensional pictures acceptable to SAP constitutes a language L. The syntactic component of SAP accepts a picture L_i and translates it to a one-dimensional string L^*_i. This one-dimensional string L^*_i is a structural description of the picture L_i. The set of all structural descriptions of L constitutes a language L*. The string L^*_i is sent to the semantic component to allow an identification or Label_i to be applied to picture L_i. This process is represented by the solid lines in Figure 1. The inverse process is the insertion of a Label_i into SAP, whereby a picture L_i is generated. This is represented by the dashed lines in Figure 1. It is interesting to note that the use of L_i or Label_i as input does not uniquely determine the other as output.

The user is able to present SAP with a syntactic string L^*_i. A syntactic metalanguage L** allows the user to present SAP with only well-formed strings. A structural description created by the user can be sent to the semantic component (solid line) to receive a Label_i or sent to the syntactic component (dashed line) to generate a picture.

The second chapter describes and defines the basic concepts which are taken from the fields of linguistics and computer science. The nature of graphics is discussed as is the difference between problems of the syntactic analysis of a natural language and problems of the syntactic analysis of pictures.

The third chapter provides a survey and contrast of the various syntax-directed pattern recognition systems which have been described in the literature.

The fourth chapter is a formal presentation of a syntax-directed language which has been designed to enable a user to describe by a one-dimensional string of symbols the two-dimensional type of pictorial data that SAP is able to consider. The strings are composed of symbols representing the geometric primitives and syntactic relations which comprise the pictorial data.

The fifth chapter describes the syntactic component of SAP. Rules are presented to combine a set of digitized points into lines and the lines into geometric symbols. A lexicon is used to obtain the syntactic relations between the geometric symbols. A syntax tree is formed and from this a linear string is derived to represent the syntactic structure of the picture.

The sixth chapter is a description of the semantic component of SAP. The syntactic string is abstracted to obtain individual figures and their basic characteristics, or Gestalt features. A semantic analysis using context sensitive rules attempts to identify the figure on the basis of its syntactic structure. Unidentified figures are then identified by context sensitive rules in terms of the syntactic structure of the scene.

The seventh chapter provides a discussion of possible further extensions of SAP and syntax-directed pattern analysis models in general. Also offered is a discussion of the advantages and disadvantages of such a model.

The eighth chapter contains a summary and conclusions of the work presented in this report.

In Appendix 9.1 is contained a description of the implementation of SAP on the LINC[1] in LAP6[2], the IBM 360/50 in LISP 1.5[3,4], and the IBM 7072 in FORTRAN. Appendix 9.2. contains a listing of the grammars and transformation rules presented throughout the report.

To provide a basis for the discussion of the following chapter, it will be assumed that graphics (or two-dimensional pictures) is some form of language. The terms graphics and pictures will be used interchangeably. Whether the two-dimensional picture language should be considered a natural, artificial, or other type of language is unclear. While no claim is made for a picture

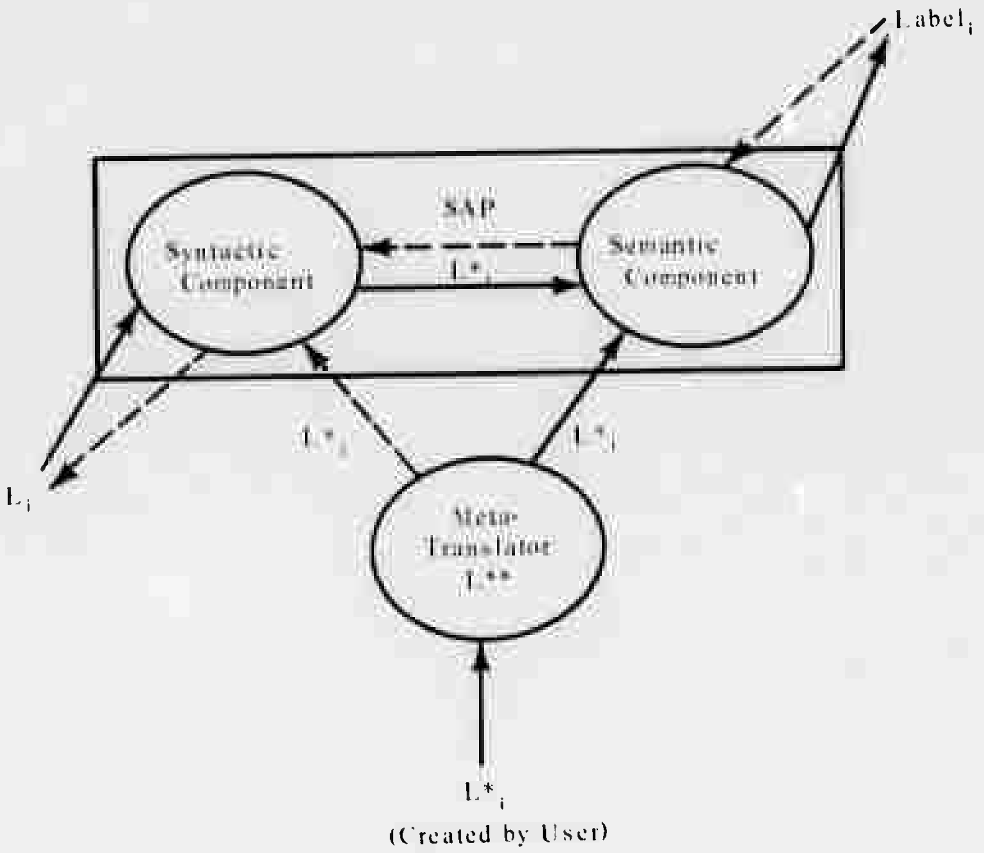


Figure 1. Outline of Flow in SAP

language as being a form of natural language, to understand better the nature of at least two-dimensional pictures, they will be described in terms of some of the considerations given to English by a structural or syntax-directed approach to natural languages.

The choice of geometric symbols as the higher level primitives was made partially as a matter of convenience and also on the basis of the desired graphic input. It was required that the scenes be rich enough in contextual information to allow our model sufficient opportunity to be tried but not overwhelmingly complex as to cause an excessive number of side tracking problems. Thus, it was decided to work with out-door type scenes as found in the country or city. In addition, it was thought best to begin work in two dimensions. In view of this, a basic set of geometric symbols were chosen. It was rather surprising the large number of sophisticated figures which could be drawn from the small set of geometric primitives. A page of these scenes is found in Figure 2. However, it should be noted that the pattern recognition of these particular scenic figures is irrelevant to the goal of indicating a general approach to the syntactic analysis of pictorial data.

The syntactic string which is processed by the pattern recognizer may be considered a data structure. This data structure allows not only the syntax of the graphic input to be concisely represented, but also allows the semantics of the scene to be obtainable. The semantics of the scene are the various meanings or recognitions that can be made in conjunction with the figures of the scene, though other semantic levels of pictorial expressions could be defined.

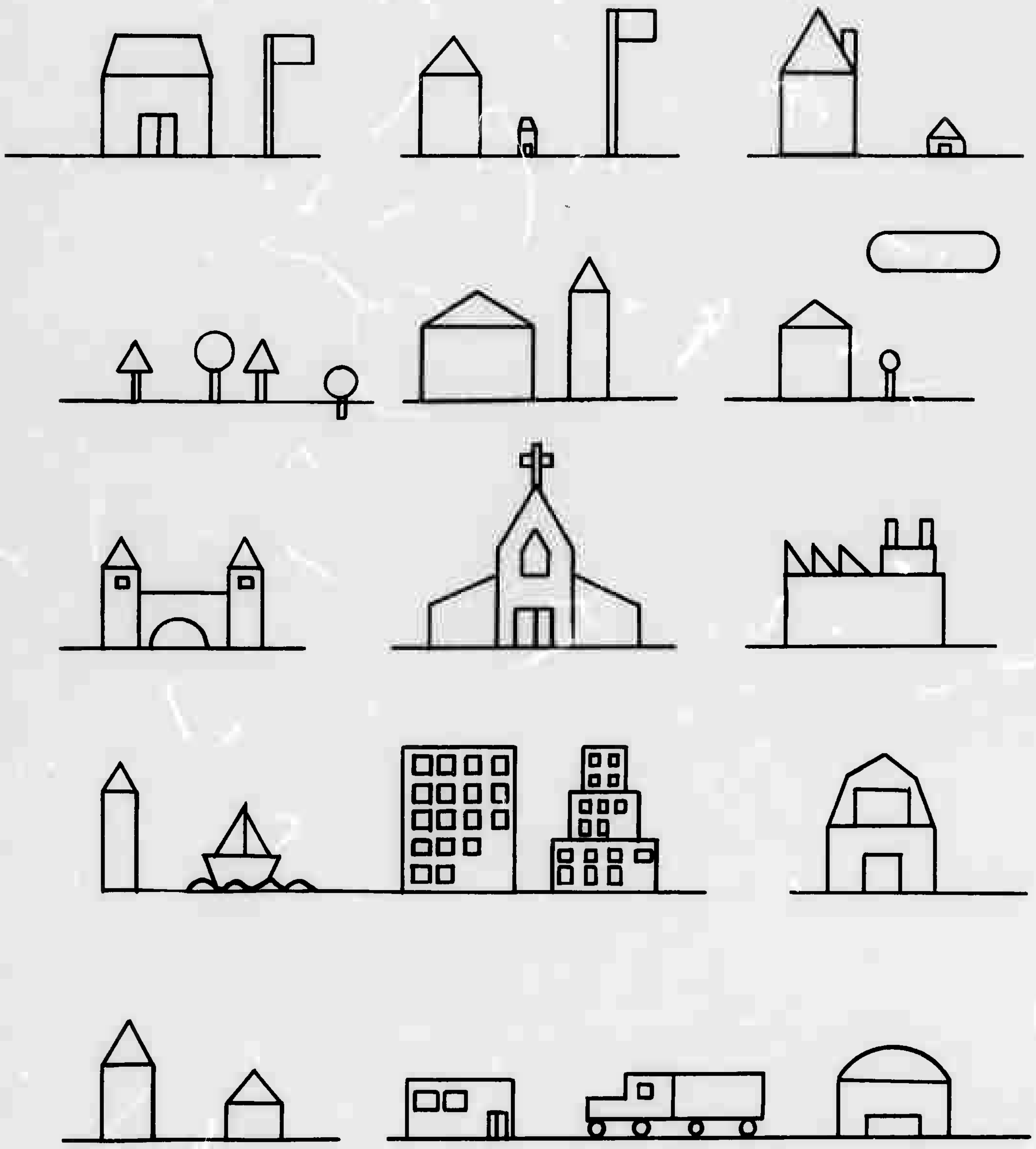


Figure 2. Examples of Scenic Input

2. BASIC CONCEPTS AND CONSIDERATIONS

2.1 SYNTAX-DIRECTED PATTERN RECOGNITION

As indicated in the introduction, syntactic analysis is meant to imply both the generation and the recognition of pictures. However, strictly speaking syntax-directed analysis refers to the recognition of patterns.

As indicated by Unger,[5] there is a distinction between pattern recognition and pattern detection.

> Pattern detection consists of examining an arbitrary set of figures and selecting those having some specified form. Pattern recognition consists of identifying a given figure which is known to belong to one of a finite set of classes.

Patterns can be considered as a large number of ordered discrete points. By giving an identification or label to a pattern essentially a many-to-one mapping is being performed. Accepting pattern recognition to be a many-to-one mapping, the difficulty in pattern recognition becomes one of determining what operations will perform this mapping. The operation which performs this mapping must obtain a set of measurements (n-tuples) which characterize the pattern. Thus, a major problem of any pattern recognition model is the choice of measurements with which it attempts to characterize the patterns which it wishes to name.

The syntax-directed method of pattern recognition essentially analyzes the patterns for connectivity and topological features. By considering the topology and geometry of the patterns, sets of n-tuples are formed, which are geometrically and topologically related. The formation of these sets is accomplished by a grammar. The grammar may be providing a syntactic or semantic analysis. Hence, a syntax-directed pattern recognition model may be represented as a set of grammars which serve as input to the computer along with the input pattern, which is to be identified. The computer then serves as a translator which operates according to the grammar rules to translate (map) from a picture L_i to its Label_i.

The input of the grammars for the syntactic and semantic analyses constitutes providing a syntactic component and a semantic component to the computer. This is shown in Figure 3. In terms of SAP, it is actually the semantic component which performs the recognition or labeling of the picture. The syntactic component provides a structural description of the picture which allows this recognition to take place.

2.2 THE PHRASE STRUCTURE GRAMMAR

A graphic or pictorial language L is considered to be a subset of the set A* of all finite arrays of symbols from an alphabet A. The language L is generated by the set of alphabetic symbols A and a set of rules for combining these symbols into a *hierarchy of constituents.* The alphabet plus the set of rules is called a grammar. Essentially, a grammar provides a description to account for observed patterns, and thus may be considered an abstraction of these common patterns. A theory of graphics would be required to choose an adequate grammar on the basis of given graphic data. While criterion used in linguistics, such as Chomsky's[6] descriptive and explanatory levels of adequacy might be valuable it certainly would be premature to propose such criterion for a starting point in a theory of graphics.

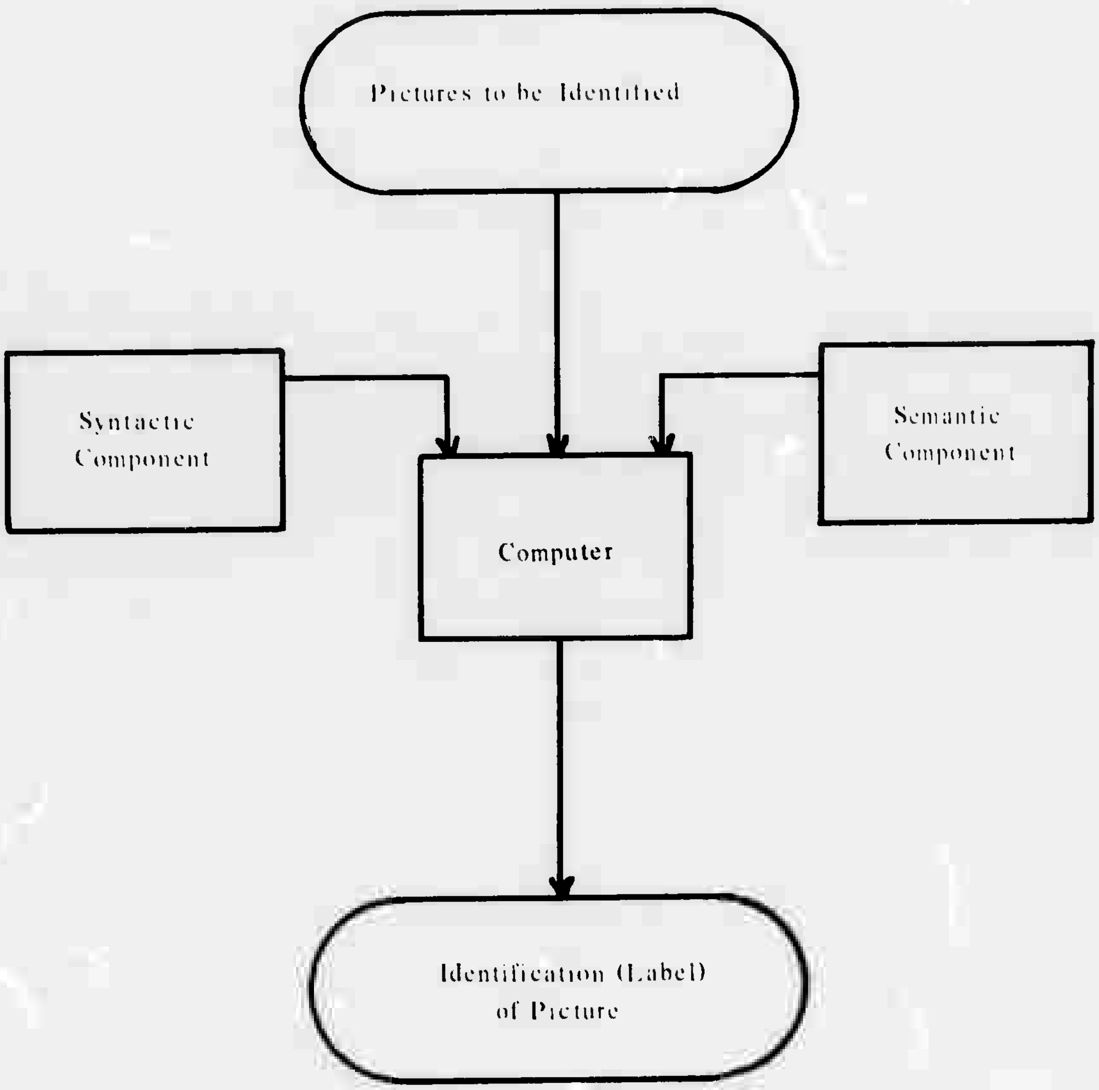


Figure 3. The Syntax-Directed Model

The type of grammar which is used to perform the analyses of the graphic or pictorial data is called a phrase structure grammar (PSG). While the components of SAP use modifications of the standard PSG, a definition of a PSG grammar is as follows:

Essentially a phrase structure grammar, K, is a finite set of productions of the form, $\phi \rightarrow \psi$ where,

(i) ψ is a nonnull string of symbols of the vocabulary V.
(ii) ϕ is a single symbol of the type called a nonterminal symbol of the vocabulary V.
(iii) There is one ϕ called the goal, S, where the goal is never a member of a ψ string.

The vocabulary V is the union of the set A of alphabetic symbols called terminals symbols and a set N of syntactic metalanguage symbols used for defining the language L and which are called nonterminal symbols.

A phrase structure grammar defines a language by forming derivatives of the goal, S. If in applying a production rule $\phi \rightarrow \psi$, the string β is said to be a direct derivative of the string α $(\alpha \Rightarrow \beta)$ if there are strings γ and δ such that $\alpha=\gamma\phi\delta$ and $\beta=\gamma\psi\delta$. The strings γ and δ may be null in the case that the production rules are context free, otherwise the rules are considered context sensitive. The operation $\alpha \overset{*}{\Rightarrow} \beta$ is defined as the case where there exist strings α_0, $\alpha_1, \ldots, \alpha_i$ such that $\alpha=\alpha_0$, $\alpha_0 \Rightarrow \alpha_1, \ldots, \alpha_{i-1} \Rightarrow \alpha_i$ and $\alpha_i=\beta$. In this case β is called a derivative of α and the sequence, $\alpha=\alpha_0 \Rightarrow \alpha_i, \ldots, \alpha_{i-1} \Rightarrow \alpha_i=\beta$ is called a derivation of β and α.

The derivatives of the goal are called sentential forms. Those sentential forms which consist only of terminal symbols are called the set of sentences (for pictorial data, the sentences are called scenes) and it is the set of scenes which comprises the language L. That is,

$$L = \{ s \mid S \overset{*}{\Rightarrow} s \text{ and } s \in A^* \}$$

2.3 PARSING STRATEGIES

The derivations of the scenes of L from the goal S are often represented by tree structures. The derivations provide a structural description (or parsing) of the scenes. The tree structures, called syntax trees, explicitly represent the structural descriptions or parsings.

The derivations of the sentences from the goal S by means of the PSG use basically one of two algorithms or parsing strategies. These are called *top-down* and *bottom-up* strategies.

1. The top-down strategy is completely goal oriented. The main goal S is chosen first. This goal chooses a set of subgoals. The subgoals hope to find a derivation of the scenes from S by substituting the right hand side of the production rules in the place of the subgoals. The substitution forms a new set of subgoals. Thus, each subgoal in turn chooses a set of subgoals. If a subgoal fails its task it is rejected and a new subgoal replaces it. It is hoped that the subgoals will eventually reach the terminal string. The top-down strategy causes infinite looping if the strings are analyzed from left-to-right and a left recursions rule occurs, that is $\phi_i \rightarrow \phi_i\psi_i$. The parsers which use a top-down strategy are sometimes called predictive, since at each step they attempt to predict the subgoals to be used to reach the terminal string.

2. The *bottom-up* strategy has only implicitly the long range goal, S. The strategy is essentially an attempt to substitute substrings which are the right hand side of production rules by their corresponding left hand side. In this manner, it is hoped that eventually the single symbol S, will be reached.

Both parsing strategies are used in SAP. The syntactic component uses a bottom-up strategy to obtain a structural description of the pictorial input. The semantic component uses a right-to-left modified bottom-up parsing to abstract the structural description provided by the syntactic component. The semantic component also uses a modified top-down strategy to assign a semantic interpretation to the figures of the scene and a bottom-up strategy to assign a semantic interpretation to the pictorial scene.

2.4 THE NATURE OF PICTORIAL DATA

One aspect of pictures which should be mentioned is the presence of pictorial universals, though as Hockett[7] points out, we do not want to invent language universals but discover them. Some of these universals are:

(i) Arbitrary configurations of pictorial data (figures) can be created at will.

(ii) The newly created figures can be considered discretely defined messages.

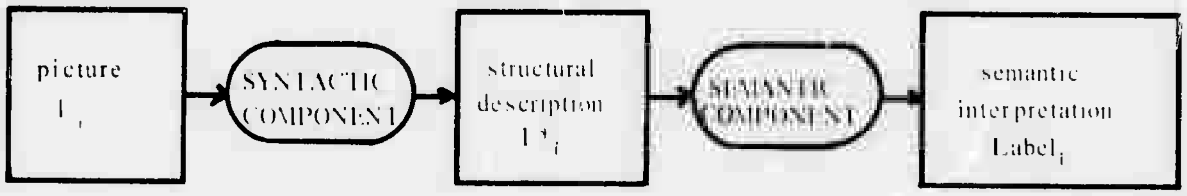
(iii) The figures may be assigned meaning independent of any physical or geometric form of the figures and also independent of the spatio-temporal coordinates of the figure.

(iv) For any non trivial graphic language, there is the possibility of ambiguous and anomalous pictorial data.

As defined by Chomsky,[8] one should note the difference between a competence and a performance model of the syntax of the language. Competence is to be considered the viewer's knowledge of graphics while performance is the viewer's actual use of this knowledge. It is the ideal viewer's intrinsic competence which is represented by a grammar. The grammar assigns to each figure a structural description which indicates how this figure is *grammatically understood* by the ideal viewer. The *acceptability* of a figure refers to the performance model and based on such factors as memory limitation it is not of present interest. The competence model is concerned with the grammaticalness of a figure.

A pictorial message goes through the same encoding and decoding processes as a linguistic message. Thus, an individual encodes a graphic message by creating it in such a manner that the message can be visually perceived. The graphic message is then decoded by either the original sender or another viewer when an attempt is made to understand the message. The message can be a single alphabetic symbol called a *primitive,* a structure composed of several primitives called a *figure,* or a structure composed of one or more figures called a *scene.*

The encoding and decoding of a graphic message is accomplished by syntactic and semantic components. It is the decoding operation, performed by the syntactic and semantic components, that is generally considered syntax-directed pattern recognition. The syntactic component parses the graphic data. The parsed graphic data is represented by a structural description (SD) which indicates the primitives comprising the graphic data and the syntactic relations between the primitives. The semantic component is highly dependent on the syntactic component and probably should be called the semantic-syntactic component. The semantic component accepts the structural description as input and provides a semantic interpretation (SI) to the graphic data. This semantic interpretation indicates whether the data is recognized, ambiguous, or anomalous.



A picture L_i may actually have more than one structural description. Thus, L^*_i represents the set of structural descriptions of L_i. A member of the set L^*_i, structural description j of picture L_i, is represented as L^*_{ij}.

Similarly, picture L_i may have more than one semantic interpretation, in which case the picture is considered ambiguous. Thus, $Label_i$ represents the set of semantic interpretations assigned to picture L_i. A particular identification or label of L_i is represented as $Label_{ij}$.

The decoding operation can be represented by the following equations:

$SD(L_i) = L^*_i$

where, $L^*_i = \{L^*_{ij}\}$ and if for L^*_{ij},

(i) max $j = 0$, then L_i is not well-formed.

(ii) max $j = 1$, then L_i has a single structural description.

(iii) max $j > 1$, then L_i has multiple structural descriptions (multiple parsings).

$SI(L^*_i) = Label_i$

where, $Label_i = L \{Label_{ij}\}$ and if for $Label_{ij}$,

(i) max $j = 0$, then L_i is anomalous.

(ii) max $j = 1$, then L_i is singularly identified.

(iii) max $j > 1$, then L_i is ambiguous.

In the remainder of the report, the subscript j will be left off unless a reference is to be made to a particular structural description or identification.

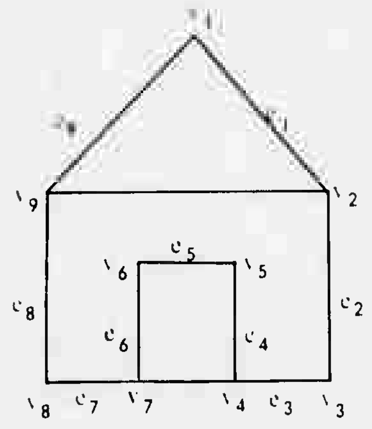
The graphic data or picture may also be considered a geometric graph. A geometric graph in two-dimensions is a set $V = \{v_i\}$ of points and a set $E = \{e_i\}$ of simple curves satisfying the following conditions:

(i) Every continuous, non-self-intersecting curve in E whose end points coincide contains exactly one point of V.

(ii) Every continuous, non-self-intersecting curve in E which joins two distinct points contains precisely two points of V, and these agree with its end points.

(iii) The curves of E have no common points, except for points of V.

Thus, a geometric graph is a geometric configuration or structure, in this case in two-dimensions, which consists of a set of points interconnected by a set of nonintersecting continuous curves. An example is shown on the following page. The fact that a figure can be considered a geometric graph will be used in Chapter Five.



2.5 LINGUISTICS DATA VERSUS PICTORIAL DATA

While the application of linguistics to graphics will be carried through the remainder of the report, it is worthwhile to note some major difficulties in syntax-directed pattern recognition which, for the most part, do not appear in linguistics.

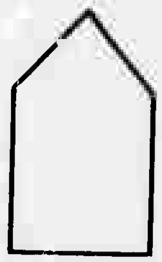
(1) A figure can be parsed from almost any direction, where theoretically each parsing readily can provide the same information. While this in itself is not a problem, since the viewer can always approach the figure from one standard direction if necessary, this flexibility indicates that restrictions must be considered because of the astronomically large number of parsings of only a slightly complex picture. As indicated earlier, the set of parsings of picture L_i is represented by L^*_i, where a particular parsing j of picture L_i is L^*_{ij}.

(2) The syntactic relations of a grammar have a hierarchy which limits the number of possible parsings of a figure into its primitives. However, for the restricted pictorial data being considered (elimination of depth) any correct parsing of the figure provides essentially the same structural description of the figure and hence, ideally, the same semantic interpretation in terms of pattern recognition.

(3) Given a parsing of a figure, in a great many cases, the semantic interpretation procedure is highly context dependent. In such a situation, a primitive cannot be semantically interpreted independent of the surrounding primitives to which it is syntactically related.

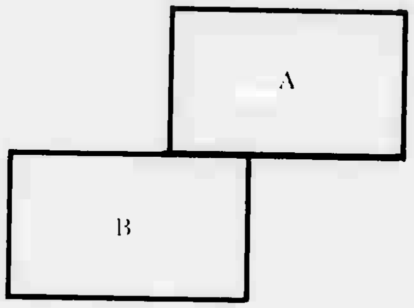
(4) The elements of the structural description and the ease with which the structural description is produced is dependent on the choice of the primitives and the syntactic relations between the primitives. However, an important point in which graphics differ from linguistics is just how the parsing is to be performed. In linguistics, the primitives or lexical items (words) are clearly distinguishable in any terminal string. Unfortunately, in graphics this is not always the case. For example, consider the simple graphic grammar composed of the two primitives, triangle and rectangle, and the single syntactic relation, *on top of*.

The figure.



is composed of a triangle on top of a rectangle with their common boundary removed. But for this figure to be considered well-formed, it must be parsed into a triangle and a rectangle, both of which are *implicitly* contained within the figure. In linguistics the lexical items are contained *explicitly*.

(5) Not only are the primitives often implicit in the pictorial data but it is also the case that the syntactic relations are always *implicit* in the data. Lexical items are concerned with only one dimension and hence are syntactically related in a string by the singular relation of *linear juxtaposition*, or the relation of *next to*. Two-dimensional pictures are concerned with juxtaposition in an infinite number of directions. The significance of the additional dimensions to juxtaposition can be seen by the example.



where A can be considered *on top of* B, but must be specified also *to the right of* or *to the left of* B to completely determine the scene.

(6) As indicated in paragraph number (4), the fact that the primitives are contained implicitly in the pictorial data is different from the occurrence of words in a natural language. However, this difference could be eliminated if the equivalent of words in linguistics are lines in figures. While this equivalence will be seen to shortly fall apart, the analogy is carried further. Both words and lines are clearly defined in their respective data. Higher level categories such as NP, VP, etc. are contained implicitly in the linguistic data just as higher level categories such as triangle, square, etc., are contained implicitly in the pictorial data. Unfortunately, the analogy collapses when the semantic aspect is considered. For while words have grammatical and semantic significance in themselves, the line has only grammatical significance in isolation from its contextural

surrounding. The same difficulty arises if words are equated to higher level constituents such as triangles, for the triangle has no semantic significance independent of its use in a specific figure.

An alternative is to consider letters of words to be equivalent to lines of a figure. In this case, both letters and lines are clearly defined in their respective data and neither have any meaning in themselves. It may even be argued that except for figures composed of one dimensional primitives (bubble tracks, etc.) lines take on no individual meaning even within their contextual environment. This is certainly the case for letters, therefore letters and lines appear to be *reasonably* equivalent. But the analogy again can go no further, for while letters form words and lines form geometric symbols it has been seen that words and geometric symbols are not equivalent in their respective languages. However, the equivalence between constituents of linguistic data and pictorial data can possibly be made on the following basis. Pictorial figures can be equated to words. The primitives which comprise the figures are equivalent to the letters which comprise the words. The lines which comprise the primitives are the same as the strokes which comprise the letters. Finally, the figures combine to form a scene as do the words combine to form a sentence. This equivalence is summarized in Table 1.

(7) A major difference in the components which perform the analyses of the graphic and linguistic data is seen by the fact that the syntactic and semantic components of pictorial analyzers are composed of essentially phrase structure grammars. The transformational movement in linguistics considers the semantic component to be composed of what Fodor and Katz[9,10] call *projection rules*. However, as pictorial analysis develops, any form of the PSG may very likely be found inadequate for semantic interpretation. It should also be pointed out that Fodor and Katz have not been having overwhelming success with their projection rules.[11]

The above problems will be discussed in greater detail in the following sections. However, the fact that the general syntactic-semantic approach is readily applicable to both linguistics and graphics leads one to believe that, within human conceptualization, pictorial and linguistic analyses are highly intermeshed. One possible major link between the languages is the Gestalt factor, which to a large extent has been overlooked in linguistic analysis. The Gestalt factor will be considered again in a later section, though it may be argued that in a discussion on the competence model, the Gestalt factor is out of place being an aspect of the performance model.

2.6 THE SYNTACTIC AND SEMANTIC COMPONENT GRAMMARS

The actual choice of a grammar in terms of the primitives and a set of relations which are able to provide a syntactic and semantic interpretation to the graphic data is a very difficult one. Assuming that a number of grammars are available, the choice as to which grammar will most efficiently and effectively process the data is more difficult that the similar problem in linguistics. If a hypothetical multilingual machine is presented with linguistic data it would first search the lexical units of the data to determine with which language it is to be dealing.

Table 1. The Equivalence of Pictorial and Linguistic Data

Pictorial Data	Linguistic Data
Scenes	Sentences
Figures	Words
Primitives	Letters
Lines	Strokes
Points	Points

The decision as to what language immediately limits the choice of grammars from which the machine can choose to further analyze the data. As indicated earlier, the primitive units in pictorial input are not so easily discernible.

However, this does indicate an approach to the problem. If a graphic processor (human or machine) is confronted with data, it will choose a grammar which seems most applicable. If this grammar does not allow a satisfactory semantic interpretation to be made, the grammar is discarded, a new grammar is chosen and the process is repeated. While it may be argued that the choice among grammars is never really made since an individual has only one large multi-leveled pictorial grammar, the argument proves nothing. Combining several small grammars into one large grammar still requires the parsing of the figures into particular primitives. Which primitives this will be requires a somewhat trial-and-error process to take place. The important factor is the development of a set or orientation. For example, suppose an individual is processing some data and finds the parsing to be reasonably easy but the semantic interpretation to be somewhat confusing. When told that what he has been calling photographs of houses and trees are really photographs of paramecium, he realizes his set orientation was wrong.

3. A SURVEY OF THE LITERATURE

The following briefly outlines some of the syntax-directed models which have been described in the literature. An extensive literature survey on syntax-directed models developed before 1966 has been made available by Jerome Feder.[12]

The work of R.L. Grimsdale[12] *et. al.* was one of the first pattern recognition programs to use a structural description of the figure to be recognized. The recognition is performed by comparing a statement describing the basic features of the pattern to be recognized to a set of statements stored in the computer which relates to named patterns. The statements describing the figure is found by a scanning process which segments the figure into *groups*, and an assembly process which obtains descriptions of the groups determines the relationships between the groups, and which compresses and codes this information to form the statement. Two implemented systems have been produced, one using a *key word* for the pattern to provide a more rapid selection of the standard pattern statement.

At the National Bureau of Standards, Russell Kirsch[14] devised one of the earlier programs which uses an immediate constituent grammar to analyze pictures of black and white triangles, squares, or circles. A limited model accepts both pictures and sentences describing the pictures as input. The sentences and pictures are parsed by phrase structure grammars and then translated into an intermediate logical language. The logical interpretation of the sentences are then tested to determine their truth value.

R. Narasimhan[15,16,17,18,19] while at the University of Illinois, had considerable success with the syntax-directed analysis of bubble chamber pictures. A program called BUBBLE SCAN performs the syntactic analysis. After forming line elements from points by a rectangular-array representation, the line elements are labeled according to their horizontal, vertical, or diagonal direction. The vertices connecting the line elements are labeled providing a labeled graph of the picture. The line elements form higher level constituents called *tracks* and the tracks form the highest level constituents on the basis of the type of constraining vertices. The entire parsing process uses a *bottom up* scheme. A second program, BUBBLE TALK, allows on-line conversation in connection with the analysis of the bubble chamber pictures. It is able to function as a complex information retrieval system in order to locate picture objects with specified attributes. Narasimhan has also dealt with the problems of noise, preprocessing, and briefly with the relation of syntactic description of pictures to the Gestalt phenomena of visual perception, though this last work appears rather inconclusive.

While the work on pattern recognition by Herbert Freeman[20,21,22,23] at New York University does not explicitly use syntax-directed analysis, it is interesting to note that the encoding process which he applies to describe an object's contour is the use of a rectangular grid as does Narasimhan. Freeman has described manipulations of the codings such as expansion and rotation. For the work on SAP, the direction grid has been formalized as a grammar for encoding a set of boundary points into labeled lines.

Some work on the grammatical formalization of handwriting has been done by Murray Eden[24,25] at MIT. Though originally developed for the generation of hand-writing, the research has developed to the stage of being used for computer recognition programs. Four basic strokes, called segments, are defined. These segments are transformed by rotation, reflection, and translation or combined to form 18 strokes whose sequences are sufficient to describe the English upper and lower case letters. The rules for collating the strokes may deal with letters or with strokes between letters.

For his master's thesis at the University of Illinois, Kenneth Breeding[26] developed a grammar to describe simple planar pictures by labeled line segments and selected vertices. The drawings are composed of straight lines oriented in either a horizontal or vertical direction. The selected vertices are those which have a degree of three. The manipulations of the strings describing the pictures is similar to, though surpassed by, the work of Freeman.

Robert S. Ledley[27,28,29,30] has done some interesting work in the syntax-directed analyses of pictures of chromosomes at the National Biomedical Research Foundation. After obtaining the boundary of an object in a photograph, the object's contour is analyzed by a grammar which has five basic curves as its terminal symbols. These curves form higher level constituents by a bottom-up parsing which determine whether or not the object is a chromosome. Ledley's earlier work in the syntactic processing of pictures was concerned with what he called *concept recognition*. One technique of concept recognition was termed *deductive reference*, which was essentially pattern recognition by a bottom-up parsing using a grammar for pictures such as cartoon-like houses. The second technique of concept recognition, *inductive inference*, uses a *General Problem Solver* approach by guessing a final goal or recognition solution of the picture and then applies a set of heuristics to reduce the difference between a syntactic description of the guessed solution and a syntactic description of the input picture to be recognized. Ledley's use of the GPS approach for an initial guess performs somewhat the same process as the abstracting technique SAP uses in the semantic component.

William E. Miller, Alan C. Shaw[31,32,33] and other members of the Computer Science Department at Stanford University and the Stanford Linear Accelerator Center are developing a system which is called a *picture calculus* to recognize and generate pictures. The picture calculus includes a picture description language (PDL), rules for manipulating pictures, and processors for the parsing and recognition of pictures.

The primitives of PDL are essentially directed line segments which have a head and a tail. The binary operators of PDL describe various possible concatenations of the primitives such as head to tail, tail to tail, etc. The unary operators of PDL, for example, reverse the tail and head of a primitive. The language has been used so far for the recognition of particle physics pictures and other graph-like structures and to permit the drawing and transformation of line drawings on a CRT. The picture recognition scheme, as used on spark chamber photographs, applies a top-down parsing analysis to a string of primitives which represent the structure of the picture. The picture generation scheme also stores the picture description as a string, which can be parsed to allow changes to be made in the picture description. Further extensions are to include a continuous transformational operator which could give the effect of motion, the development of PDL to utilize what might be considered higher level topological concepts, such as *contained within*, *adjacent*, and *above*, and the consolidation of the recognition and generation schemes to facilitate learning by the system.

A somewhat different syntactic approach has been taken by William Martin[34] and Robert Anderson.[35,36] Martin, at MIT, is concerned with taking a mathematical expression which is stored as a tree structure and creating a visual display of the expression. Each symbol is expressed in a grammar by a special form, each form is then inscribed by a dimensioned rectangle. The dimensioned rectangles combine to create a higher level dimensioned rectangle which is then centered on the cathode-ray tube and the contained expression displayed. Anderson, for his Ph.D. dissertation, has provided a grammar for the recognition of various mathematical expressions which are displayed in two-dimensions. The model partitions a displayed configuration into syntact c subcategories which are inscribed in dimensioned rectangles.

The rectangles are, in turn, positionally related. However, as Anderson[36] points out, he is not working from a tree structure to a display as Martin, but, from a display to a tree structure.

M.B. Clowes,[37,38] at CSIRO in Australia, has developed an interesting grammar for describing numerals in terms of the contiguous edges which form their boundary. Clowes has also related this work to similar research in physiology and psychology.

These models all use syntax-directed analysis for at least some aspect of their total analysis. A comparison of the various models is made difficult, because one uncontrolled factor which, unfortunately plays a large role in determining the overall models' appearance is the type of input patterns being considered. As pointed out in the previous chapter, the nature of the data determines the type of primitives and syntactic relations which comprise the data.

For example, the fact that Narasimhan considered figures composed of only one-dimensional constituents (straight lines) eliminated an interesting semantic problem. When a line was found in Narasimhan's bubble chamber picture, it was considered a track. Hence, the line is able to receive at least a partial semantic interpretation immediately. However, in considering the primitives used by SAP, they generally cannot receive a semantic interpretation independent of their context.

A further simplification in analyzing figures of bubble chamber tracks is that the syntactic relation between them is singularly concatenation. This allows the syntactic description to be directly represented as a one-dimensional string and processed by a grammar for semantic interpretation. However, if the primitives are two-dimensional, as in the case of SAP, a syntactic analysis must first process the figure to obtain some form of explicit representation of the topological relations between the primitives. This representation may then be operated upon the grammar to receive a semantic interpretation. However, it should be noted that both Narasimhan's and Miller's work has proved very satisfactory for the pattern recognition of bubble chamber tracks. Similarly, Ledley's pattern recognition of chromosomes considers only the syntactic relation of concatenation. In all cases the concept of *contiguity* is the determining factor. The difference in approaches essentially determines the number of different values under which contiguity is considered. It is interesting that Clowes does not consider numerals to be composed of one-dimensional line segments but by using a grammar which describes a figure by the line segments which form the figure's contour, he uses only the relationship of *next to*.

The use of figures whose primitives are one-dimensional lines which require only the syntactic relation of concatenation does not necessarily eliminate the problem of multiple parsings of the figure. As is shown in Chapter 5, a two-dimensional figure may have a very large number of different parsings. This is mainly because the topological relations which have been chosen are associative when serving as operators in a syntactic description. Anderson avoids this problem by not explicitly using any topological relations in the syntactic description.

While Grimsdale uses topological relations between segments of alphanumeric figures, the semantic interpretation is accomplished by phrase matching techniques. The use of a grammar would be a strong addition to his model. Though Grimsdale's results must be taken *cum grano salis* (with a grain of salt), as Uhr[39] states, Grimsdale's work "is generally accepted as being one of the most powerful and – intuitively and psychologically – satisfying of pattern recognition programs."

Breeding's consideration of figures composed of horizontal and vertical lines does no more than provide a notation for such figures. Its use in pattern recognition does not appear promising. Kirsch's work is interesting and apparently has been carried further in the direction of the syntactic analysis of biological images.[40]

Anderson's approach of syntactically describing a picture in terms of its components and pattern recognizing the picture by the positioning of coordinates should apply to most of the two-dimensional pictures which SAP considers. His method would possibly have difficulty with some of the structures because the relative size of spacing between constituents is not tested. The fact that his syntactic rules for pattern recognition are not in the form of a phrase structure grammar is of no importance and that they do allow *fine tuning* of the recognition process is highly advantageous, but by not explicitly considering the syntactic relations between the primitives would cause his model to be cumbersome if used in a man machine system. Narasimhan's BUBBLE TRACK allows for a reasonably interactive system. Miller's Picture Description Language is an approach to this problem though the lack of topological relations is a limiting factor. Miller[32] points this out in stating that the picture calculus "is not very convenient for describing complex topological concepts without inclusion of concept recognizers."

While a phrase structure grammar in linguistics can serve as either a generator or recognizer of sentences, in two dimensions a single syntactic relation between two constituents does not completely determine their position. To get by this difficulty, SAP requires the syntactic relations to have arguments which further position the primitives which they relate. This will be described in detail in the next chapter. The one-dimensional figures of Narasimhan's bubble chamber tracks again cause no difficulty in this regard. Miller and Shaw also appear to be able to get by such problems by using a powerful and descriptive set of syntactic operators.

A source of information which so far has been used only sparingly in linguistics and character recognition is contextual information between figures. This will be discussed in Chapter 6. Suffice it to say, that contextual analysis is a significant source of information which has not been considered to any extent by pattern recognition models. Ledley uses contextual information [illegible] in an aspect of the chromosome analysis, while the remaining syntax-directed models do not seem to use any facet of contextual information. In some models this may be due to the fact that pictures such as bubble chamber tracks and mathematical expressions do not contain a high degree of contextual information.

In summary, the work of Narasimhan, Ledley, and Miller is similar to SAP in the use of a grammar to obtain a syntactic description of the figure. The notion of contiguity plays an important role in defining the syntactic relations, though SAP provides additional processing to obtain *higher level* syntactic relations while for the most part they need only be concerned with the singular relation of *linear juxtaposition*. In addition, their use of a grammar to obtain a semantic interpretation of the figure differs from the manner in which SAP uses a grammar. This is because of the constraints which exist between the constituents of the figures. The phrase structure grammar for Narasimhan's bubble chamber tracks and Ledley's chromosomes are context free as is Narasimhan's[19] grammar for the pattern recognition of alphabetic characters. In Chapter 6 is a description of how the constraints which can exist between either constituents of a figure or between figures of a scene may be taken into account by a modified context sensitive phrase structure grammar.

The next chapter describes a language L* which can serve as a meta language for describing a two-dimensional language L, such as the figures illustrated in the introduction.

4. A SYNTAX-ORIENTED LANGUAGE L*

4.1 THE LANGUAGE

The set of pictures which serve as input to SAP are considered to comprise a two-dimensional language L. As indicated in Figure 1, the syntactic component of SAP translates a member of L (a two-dimensional picture L_i) to a set of one-dimensional strings of symbols, L^*_i. A particular string j is L^*_{ij}. The set of all strings which result from the translation of the set L of two-dimensional pictures defines a language L*. Hence, the syntactic component of SAP performs a translation of a two-dimensional language to a one-dimensional language.

In transforming a two-dimensional language to a one-dimensional language certain operations must take place. This is because, for a one-dimensional pattern to completely determine a two-dimensional pattern, the topological features which are implicit in the two-dimensional pattern must be made explicit in the one-dimensional pattern. Thus the one-dimensional string L^*_i which results from the translation of the picture L_i, describes the topological or syntactic features of L_i. Because the strings of symbols L* describe the syntax of the pictures of L,L* may be considered a metalanguage. The set of strings L_i* is called the parsings or structural description (SD) of the picture L_i. Thus, $SD(L_i)=L^*_i$. The set of strings L^*_i is sent to the semantic component of SAP to receive a semantic interpretation.

This chapter presents the syntax of the language L*. The specification of the syntax of L* is described in a language L**. The production rules which describe an algorithm for recognizing a member of L* are the statements of L**. Thus, L** is a syntactic metalanguage relative to L*, and a meta-metalanguage relative to L.

As indicated in the previous two paragraphs, the set of strings L^*_i representing the structural description of a picture L_i may be obtained from the syntactic component of SAP. However, an alternative is for a user to provide SAP with a string L^*_{ij}. This allows a user to send to the semantic component of SAP the structural description of a picture without having the actual picture. In terms of picture recognition, this provides a means of testing the semantic component independent of the syntactic component. In terms of picture generation, a user can send to the syntactic component of SAP the structural description. This allows the picture to be generated without having the actual label of the picture. Thus, the syntactic component can be tested and operated independent of the semantic component. Of course, the user can send his string to both components, generating a picture from the descriptive string and obtaining a label or identification for the generated picture.

There is a little doubt that the facility of man/machine interaction is an important factor in the development and ultimate use of a model such as SAP. For this reason, the syntax of L* which recognizes the well-formed strings which the user might send to SAP is presented in this chapter in its entirety. The process by which the syntactic component of SAP translates a picture L_i into a string or parsing L^*_i will be described in the next chapter.

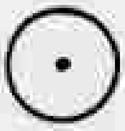
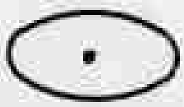
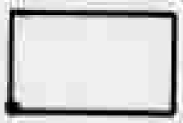
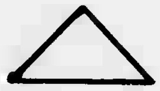
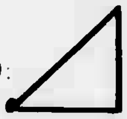
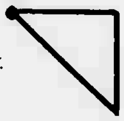
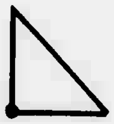
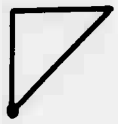
Formally, the language L* is defined by a phrase structure grammar K2* which is represented using the Backus–Naur Form (BNF). The language L* is a subset of the set of all finite strings of symbols from the alphabet A. K2* is a 6-tuple, (P,S,F,N,R,SS). The alphabet A is the union of the sets of terminal symbols P, S, and F.

The vocabulary of the language L** is a union of the sets P,S,F, and N. The set P is a set of terminal symbols which represent the primitive geometric symbols of the language L.

The set S is a set of terminal symbols which are the names of the syntactic relations which can occur between the primitive symbols. The set Γ is a set of functions which are the terminal symbols which use the primitives and syntactic relations as arguments. The set N is a set of nonterminal or category symbols of L**. The set R is a set of production rules which determine how the syntactic strings of L* are to be formed. SS is the goal of the language, a syntactic string.

4.2 THE PRIMITIVES

It was stated earlier that there is a great deal of freedom in choosing the set of primitives of the language. However, it is obvious that some primitives will represent certain pictorial data better than others. For the present work, the following set of geometric primitives have been chosen:

1. Circle	(C)	
2. Ellipse	(E):	
3. Rectangle	(R).	
4. Isosceles triangle	(TI):	
5. Right triangle	(TR):	
6. Right triangle down	(TRD):	
7. Left triangle	(TL):	
8. Left triangle down	(TLD):	

Thus, P = {C, E, R, TI, TR, TRD, TL, TLD}. The difference between a right triangle and a left triangle is the location of the right angle of the triangle.

The entire set of primitives listed above are completely determined by specifying their height, width, and a reference point on the primitive. The circle and ellipse have their center as their reference point while the remainder of the primitives have their lower left hand vertex as their reference point. The reference point is indicated by a large dot on the primitives shown on the previous page. In addition, each primitive of a given type in a syntactic string must be numbered to distinguish it from the other primitives of the same type in the syntactic string.

The general form of a primitive is defined by the syntax on this page. For example, rectangle number 3 with a width of 4 and a height of 10 would be written as,

R(3,4,10)

A circle should have the same horizontal and vertical dimension. However, it is the horizontal dimension that is used to determine the diameter of the circle.

```
           <name cir > ::=  C
         <name ellip > ::=  E
          <name rect > ::=  R
          <name isos > ::=  TI
        <name rt tri > ::=  TR
    <name rt tridown > ::=  TRD
       <name lft tri > ::=  TL
   <name lft tridown > ::=  TLD
              <zero > ::=  0
            <number > ::=  1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
            <integer > ::=  <number> | <zero> | <number><integer
            <numtype > ::=  <integer>
            <numtype > ::=  <integer>
<horizontal dimension > ::=  <integer>
  <vertical dimension > ::=  <integer>
  <primitive argument > ::=  <numtype>,<horizontal dimension>,
                                 <vertical dimension>
             <circle > ::=  <name cir>(<primitive argument>)
            <ellipse > ::=  <name ellip>(<primitive argument>)
          <rectangle > ::=  <name rect>(<primitive argument>)
  <isosceles triangle > ::=  <name isos>(<primitive argument>)
     <right triangle > ::=  <name rt tri>(<primitive argument>)
 <right triangle down > ::=  <name rt tridown>(<primitive argument>)
      <left triangle > ::=  <name lft tri>(<primitive argument>)
  <left triangle down > ::=  <name lft tridown>(<primitive argument>)
```

A figure represented by a syntactic string need have only one primitive in the string referenced in order to position all primitives and thus the figure in the field of view. A referenced primitive has a somewhat different form than that indicated above, the x and y coordinates of the reference point being arguments of the primitive. The referenced primitive may be defined as,

```
                  <xcoord> ::= <integer>
                  <ycoord> ::= <integer>  ,
                   <refpt> ::= <xcoord>,<ycoord>
<reference primitive argument> ::= <zero>,<horizontal dimension>,
                                  <vertical dimension>,<refpt>
```

And thus added to the definition of <rectangle>, etc., is,

```
               <circle> ::= <name cir>(<reference primitive argument>)
              <ellipse> ::= <name ellip>(<reference primitive argument>)
            <rectangle> ::= <name rect>(<reference primitive argument>)
   <isosceles triangle> ::= <name isos>(<reference primitive argument>)
       <right triangle> ::= <name rt tri>(<reference primitive argument>)
  <right triangle down> ::= <name rt tridown>(<reference primitive argument>)
        <left triangle> ::= <name lft tri>(<reference primitive argument>)
   <left triangle down> ::= <name lft tridown>(<reference primitive argument>)
            <primitive> ::= <rectangle>|<isosceles triangle>|
                            <right triangle>|<right triangle down>|
                            <left triangle>|<left triangle down>|
                            <circle>|<ellipse>
```

For example, a reference circle of diameter 15, and reference at (11,47) would be written as,

C(0,15,15,11,47)

4.3 THE SYNTACTIC RELATIONS

As in the case of the primitives, the choice of the syntactic relations are also somewhat arbitrary. However, also as in the case of the primitives, an optimal or perhaps even feasible analysis of the pictorial input necessitates considerable thought to be given as to the choice of grammar to be used. The choice of the grammar for a graphic language depends to a large extent on the syntactic relations desired. While it is possible that at a later date some formal criterion for choosing a graphic grammar can be presented, the best that can be said for the present is that intuitively, it appears that the higher level primitives should be chosen before the syntactic relations if the two choices can be made separately.

For use in the present research a number of syntactic relations were tried which would be meaningful in terms of the defined set of geometric primitives. The final choice of relations has been extremely successful while providing a rather simple syntax. The relations turned out to be similar to those suggested by Ledley[29] in his brief study of the syntactic representation of picture expressions. The syntactic relations are of the form *Ri(...), where the arguments contained within the parentheses will be discussed shortly and i=1,...,6. The present set of binary relations consists of:

(i) X,Y, *R1 (...) = X is on top of Y.

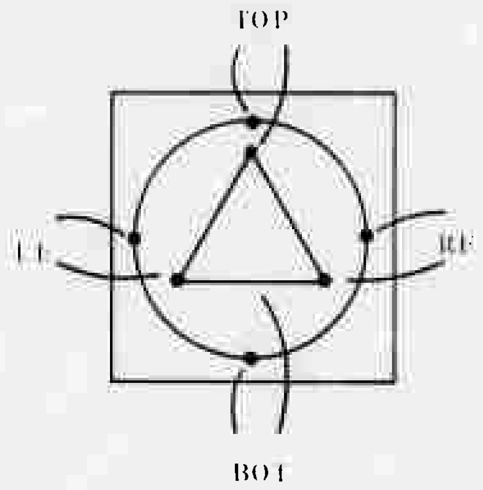
(ii) X,Y, *R2 (...) = X is under Y.
(iii) X,Y, *R3 (...) = X is to the right of Y.
(iv) X,Y, *R4 (...) = X is to the left of Y.
(v) X,Y, *R5 (...) = X is contained within Y.
(vi) X,Y, *R6 (...) = X contains Y.

or where the primitives, X and Y, can be considered operands and the relation *Ri is an operator. Thus, S = }*R1,*R2,*R3,*R4,*R5,*R6{.

The arguments of the relations *R1, *R2, *R3, and *R4 contain the following information.

(1) The magnitude or distance of the syntactic relation (MSR). (e.g., MSR=0 means the primitives are touching,...).
NOTE: MSR is optional if 0.

(2) The *Ri of the syntactic relation defines the position between primitive X and primitive Y is only one dimension (horizontal or vertical). To define the relative position between X and Y in the other dimensions (called the *secondary position*) the edges of the primitives are used (e.g., TOP=top, BOT=bottom, LE=left side, RE=right side, HC=horizontal center line, VC=vertical center line).
NOTE: Optional if HC or VC.
The edges of some of the geometric primitives in use are defined as:



Thus, the edge of a triangle may be a single point, the vertex of a triangle.

(3) The magnitude or distance of the secondary position (MSP) between the edges of the primitives and defined by the arguments of the syntactic relation must be determined.
NOTE: MSP is optional if 0.

(4) The secondary position between X and Y defined in the arguments of the syntactic relation must also be given a direction. (e.g., UP=up, DN=down, LT=left, RT=right).
NOTE. optional if (a) MSP=0, or, (b) if any of the following combinations of edges of the primitives and the direction of the

relative position are used. (LE,RT); (RE,LT); (BOT,UP), (TOP, DN).

The syntactic relations *R1, *R2, *R3, and *R4 are defined on the following page.

Examples in using the syntactic relations are:

X,Y,*R1(2,RE3LT) = X is 2 units above Y with the right edge of X 3 units to the left of the right edge of Y.

X,Y,*R1(2,RE3) = same as above with options used.

X,Y,*R3(0,BOT1UP) = X is to the immediate right of Y with the bottom edge X 1 un up from the bottom edge of Y.

X,Y,*R3(BOT1) = same as above with options used.

```
                <name sr> ::= *R
                    <msr> ::= <integer>
                   <vedg> ::= TOP | BOT | HC
                   <hedg> ::= LE | RE | VC
                    <msp> ::= <integer>
                   <vdir> ::= UP | DN
                   <hdir> ::= LT | RT
                <vrelpos> ::= <vedg><msp><vdir>
                <hrelpos> ::= <hedg><msp><hdir>
                 <argvsr> ::= <msr>,<hrelpos>
                 <arghsr> ::= <msr>,<vrelpos>
    <vertical relation 1> ::= <name sr>1(<argvsr>)
    <vertical relation 2> ::= <name sr>2(<argvsr>)
  <horizontal relation 3> ::= <name sr>3(<arghsr>)
  <horizontal relation 4> ::= <name sr>4(<arghsr>)
      <vertical relation> ::= <vertical relation 1> |
                                 <vertical relation 2>
    <horizontal relation> ::= <horizontal relation 1> |
                                 <horizontal relation 2>
   <directional relation> ::= <vertical relation> |
                                 <horizontal relation>
```

The arguments of syntactical relations *R5 and *R6 differ from the other relations in that an additional argument or constraint is needed to completely specify the *contained within* relationship. *R5 and *R6 are defined as follows.

```
                      <argcwsr> ::= <zero>,<vrelpos>,<hrelpos> |
                                       <zero>,<hrelpos>,<vrelpos>
  <contained within relation 5> ::= <name sr>5(<argcwsr>)
  <contained within relation 6> ::= <name sr>6(<argcwsr>)
    <contained within relation> ::= <contained within relation 5> |
                                       <contained within relation 6>
```

An example of the use of the *R5 relation is X,Y,*R5(0,LE1RT,TOP2DN) which says primitive X is contained within primitive Y and the left edge of X is 1 unit to the right of the left edge of Y and the top of X is 2 units down from the top of Y.

Thus,

<syntactic relation >::= <vertical relation> |
<horizontal relation> |
<contained within relation>

The *contained within* relation is actually a combination of a *horizontal relation* and *vertical relation.* Two arguments are required for the *contained within* relation since *R5 and *R6 convey no information while in the case of *Ri for i=1,...,4 the i specifies a particular dimension. Thus, for the first four relations, only one directional argument is required. Though it can be replaced by a combination of a *horizontal* and a *vertical* relation, the *contained within* relation is available to provide easier expressability by the user but more so it is needed to maintain the normalized form of the syntactic string. This normalized form requires two constituents to be related by a single binary relation. To use a *horizontal* relation and a *vertical* relation in the place of a single *contained within* relation would not conform to the normalized form. That is, the normalized form would be destroyed if two primitives, where one primitive is contained within the other, would be related by two binary relations (a *horizontal* relation and a *vertical* relation) rather than by a single binary relation (a *contained within* relation).

The following list represents the properties of the syntactic relations defined in this chapter:

(1) The entire set of relations is *irreflexive.* That is, a binary relation *Ri is irreflexive if there is no X such that (X,X,*Ri) .

(2) The entire set of relations is *unsymmetric.* That is, a binary relation *Ri is unsymmetric if, for any pair of elements X and Y for which (X,Y,*Ri) it is necessarily the case that $\sim$(Y,X,*Ri), where $\sim$ is read as *not.*

(3) The entire set of relations is *transitive.* That is, a binary relation *Ri is transitive if, for any X,Y, and Z, given (X,Y,*Ri) and (Y,Z,*Ri) implies (X,Z,*Ri).

Because the relations are irreflexive, unsymmetric, and transitive they are called proper inequality relations.

If the *Ri previously defined is considered as a binary operation rather than a binary relation the following properties hold.

(1) The entire set of operations is *noncommutative.* That is, for any X and Y, (X,Y,*Ri) $\neq$ (Y,X,*Ri).

(2) The entire set of operations is *associative.* That is, for any X, Y, and Z, (X,Y,*Ri,Z,*Ri) = (X,Y,Z,*Ri,*Ri).

(3) The entire set of operations is *closed.* That is, for every choice of elements X and Y in the set, (X,Y,*Ri) is also the set.

The general form of the syntactic relations is *Ri(n_1,n_2,n_3). The arguments of the relation have been defined by the syntax as:

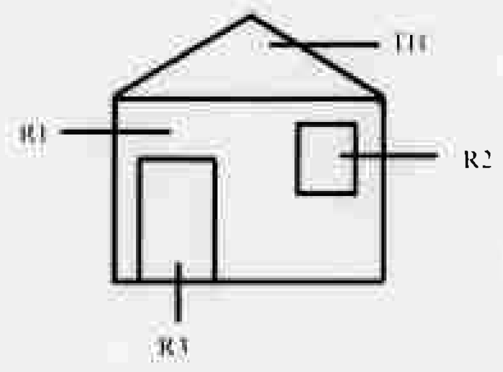
(1) n_1 is an integer providing the magnitude of the relation.
(2) n_2 is string of letters and an integer to indicate the secondary positioning.
(3) n_3 is also a string of letters and an integer to indicate secondary positioning for a *contained within* relation. The argument is null and hence absent for the *horizontal* and *vertical* relations.

The syntactic strings formed by the primitives and syntactic relations have a normalized form of a Reverse Polish ordering. This notation, invented by the Polish philosopher Lukasiewicz in 1921, has been used extensively in compiling computer languages because it makes the operators and operands in a syntactic string available at the precise moment they are required in the compilation. Early Reverse Polish requires that the operator immediately follow its operands, eliminating the need for constituent grouping by parentheses. Thus, a segment of the syntactic string can be defined as,

<sss>:: = <primitive><primitive><syntactic relation>
<sss><primitive><syntactic relation>
<primitive><sss><syntactic relation>
<sss><sss><syntactic relation>

where <sss> forms higher level constituents of the language. For example, considering the syntactic relations and primitives without their arguments, a house could be syntactically represented as:

HOUSE:

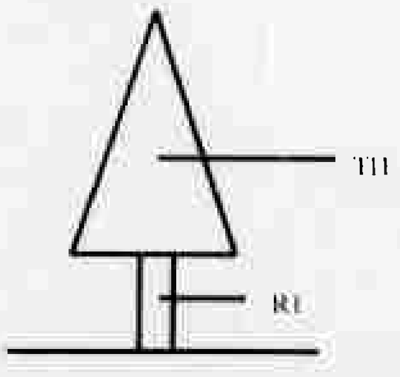


T1.R2.R3, *R3.R1.*R5, *R1

where the constituent groupings are actually, (T11.((R2.R3.*R3).R1.*R5).*R1). It should be noted that while there is more than one possibility correct parsing for a single graphic input, at this time no one parsing appears to provide any more or less information than any other parsing. This is an advantage over natural language analysis, since it does not appear necessary to obtain all possible parsings of the source language statement (graphic in this case) to obtain the full semantic content of the statement.

Using a linear scale of 1 unit approximately equal to $\frac{1}{8}$", some further examples of the use of syntactic strings to represent graphic input are:

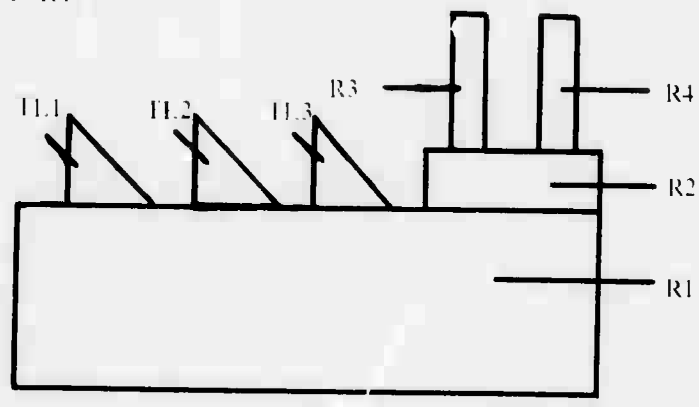
TREE:



T1(1,6,12,).R(1,2,5,).*R1(0,VCOLT)
or with the use of options,

T1(1,6,12).R(1,2,5).*R1 since the vertical center lines of the two primitives are aligned.

FACTORY



TL(1,3,3),TL(2,3,3),*R4(1,BOTOUP),TL(3,3,3),*R4(1,BOTOUP),R(3,2,6),R(4,2,6),
*R4(2,BOTOUP),R(2,8,4),*R1(0,LE1RT),*R4(1,BOTOUP),R(1,22,7),*R1(0,LE2RT)

or with the use of options,

TL(1,3,3),TL(2,3,3),*R4(1,BOT),TL(3,3,3),*R4(1,BOT),R(3,2,6),R(4,2,6),*R4(2,BOT),
R(2,8,4),*R1(LE1),*R4(1,BOT),R(1,22,7),*R1(LE2)

In the next chapter a procedure is presented for testing the well-formedness of a syntactic string in Reverse Polish form.

4.4 THE SYNTACTIC FUNCTIONS

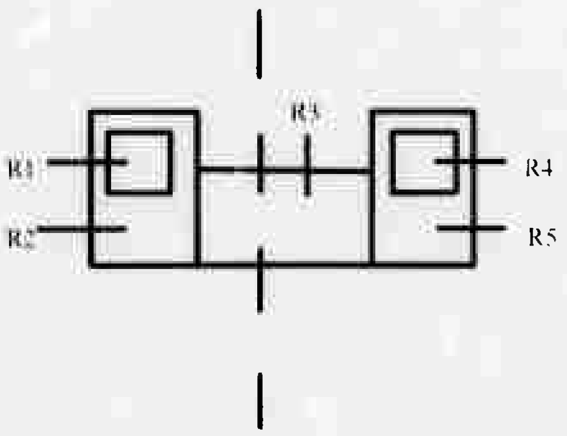
Some other relations of the graphic language might best be considered as unary and binary functions which operate on segments of the syntactic string. Some of these are:

HORIZONTAL OR VERTICAL SYMMETRY:

This is unary function which operates on a segment of a syntactic string to generate its corresponding symmetrical syntactic string. The horizontal symmetry can be either right or left and the vertical symmetry can be up or down. The symmetry functions may be defined as,

<hsym> ::= *HSYM
<vsym> ::= *VSYM
<symf> ::= <hsym><hdir> | <vsym><vdir>

For example, the building drawn below can be represented by the following syntactic string:



R(1,2,2),R(2,4,6),*R5(TOP1),
R(3,8,4),*R4(BOT),R(4,2,2),
R(5,4,6),*R5(TOP1),*R4(BOT)

using the horizontal symmetry function, the syntactic string becomes,

R(1,2,2),R(2,4,6),*R5(TOP1),R(3,4,4),*R4(BOT),*HSYMR.

To produce the remainder of the syntactic string internally, the horizontal symmetry function performs the following transformations on the syntactic relations upon which it operates:

*R1	transforms to	*R1	TOP	transforms to	TOP
*R2	⟶	*R2	BOT	⟶	BOT
*R3	⟶	*R4	LE	⟶	RE
*R4	⟶	*R3	RE	⟶	LE
*R5	⟶	*R5	HC	⟶	HC
*R6	⟶	*R6	VC	⟶	VC

The vertical symmetry function, when operating upon a syntactic string, performs the following transformations:

*R1	transforms to	*R2	TOP	transforms to	BOT
*R2	⟶	*R1	BOT	⟶	TOP
*R3	⟶	*R3	LE	⟶	LE
*R4	⟶	*R4	RE	⟶	RE
*R5	⟶	*R5	HC	⟶	HC
*R6	⟶	*R6	VC	⟶	VC

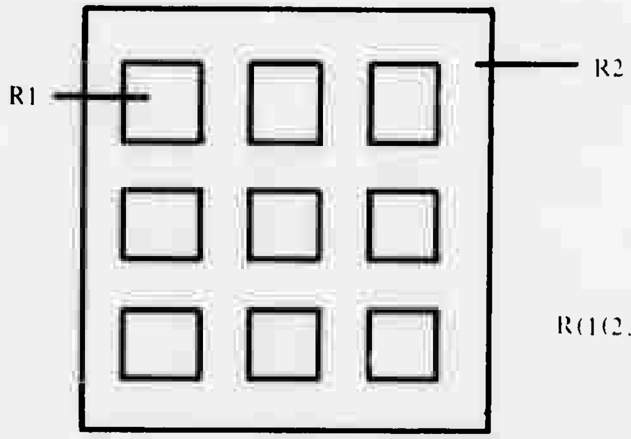
REPEAT SEGMENT:

This is a unary function which operates on a segment of a syntactic string to generate a replication of the segment. The first numeric value (MBR) indicates the magnitude of the distance between the replications where 0 is optional and indicates touching. As defined earlier (HDIR) and (VDIR) are the direction in which the replications are to be taken, used in this case along the horizontal or vertical line. The second numeric value (NUR) indicates the number of replications to be made.

The repeat function may be defined as,

```
<rp> ::= *RP
<mbr> ::= <integer>
<nur> ::= <integer>
<rpf> ::= <rp><mbr><hdir><nur> | <rp><mbr><vdir><nur>
```

For example,



R(1(2,2),*RP1R12,*RP1DN2,R(2,14,14),*R5

ENLARGE OR REDUCE SEGMENT:

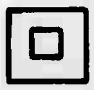
This is a unary function which operates on a segment of a syntactic string to regenerate the syntactic string with its various measurements changed by a factor indicated.

This function may be defined as,

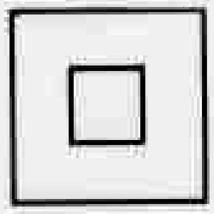
<els >::= *ELS
<rds >::= *RDS
<elrdf >::= <els><num> | <rds><num>

For example,

R(1,1,1).R(2,3,3).*R5.*ELS2



becomes



R(1,2,2),R(2,6,6),*R5

DEFINE SEGMENT:

This is a binary function which allows a dummy variable, DEFi (i=0,...) to be used in place of a segment of the syntactic string. The segment of the syntactic string which DEFi represents is defined as that portion of the string which follows DEFi until the first *DEFS is found. The integer i names the particular segment of the syntactic string which is being represented.

The define function may be defined as,

<defs >::= *DEFS
<defi >::= *DEF<integer>
<defsf >::= <defi><... ><defs>

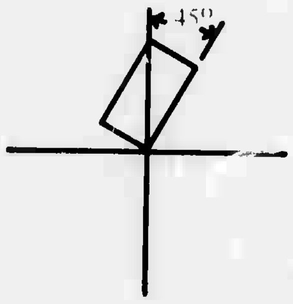
ROTATE ABOUT REFERENCE POINT:

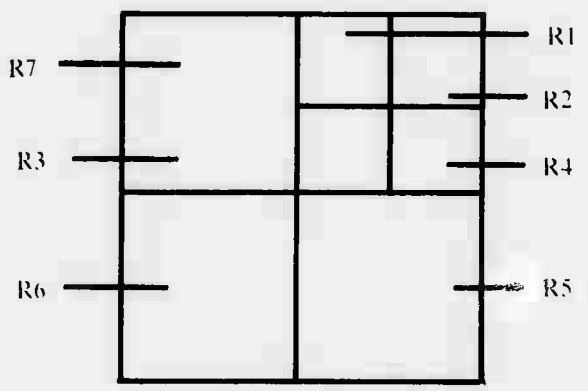
This is a unary function which is used to rotate primitives clockwise from the vertical around their reference point. The numeric value of the function indicates the number of degrees to rotate. The rotate function may be defined as,

<rote >::= *ROTE
<rotef >::= <rote><integer>

For example,

R(1,4,8).*ROTE45





R(1,4,4),R(2,4,4),*R4,R(3,4,4),R(4,4,4),*R4,*R1,R(7,8,8),*R3(BOT)
R(6,8,6),R(5,8,8),*R4(BOT),*R1(LE)
or
DEF1,R(1,4,4),*RPRT1,*RPDN1,*DEFS,DEF1,*ELS2,*R5(TOP,RE)

Figure 4. An Example of a Complex Figure Defined by Syntactic Functions

A final example of the syntactic functions in Figure 4 indicates some of the arbitrarily complex figures which can be created. While many more functions can be readily created, for those functions presented in this chapter, the syntactic function can be defined as,

<unary syntactic function>::= <symf> | <rpf> | <elrdf> | <rotef>
<binary syntactic function >::= <defsf>

Thus, to the definition of the segment of the syntactic string must be added,

<sss>=::<sss><unary syntactic function> | DEFi <sss>DEFS

The goal of the language is syntactic strings. Any segment of a syntactic string is also a syntactic string. Those segments of the syntactic string which are able to receive a semantic interpretation are considered *meaningful*. Other syntactic strings are anomalous. But in any case, the syntactic string, as long as it is well-formed Reverse Polish string, can be displayed graphically. Thus,

<ss>::=<sss>

The syntactic relations defined by the previous syntax are actually redundant. That is, only *R1. *R3 and *R5 are really needed to have the same descriptive capabilities as all six relations. Thus, SAP normalizes the syntactic string, by converting, for example, the *R2 to *R1 with the necessary manipulations of the syntactic string. Similarly, the functions are removed from the string before the string is processed, the functions being replaced by the corresponding segment of the syntactic string.

The normalized syntactic string will also be the result of the syntactic component of SAP for a given picture. The syntactic string is a structural description of the picture and once obtained, is operated upon by the semantic component to apply a semantic interpretation to it. If a semantic interpretation can be applied to the structural description then SAP has recognized the pictorial input.

The syntactic string is produced in SAP by its syntactic component. This component and the analyses it uses are described in the next chapter.

5. THE SYNTACTIC COMPONENT OF SAP

5.1 AN OUTLINE OF THE SYNTACTIC COMPONENT

The syntactic component operates on a two-dimensional picture L_i to provide a set of structural descriptions L^*_i. The structural descriptions are the same one dimensional strings of symbols described in the previous chapter. Thus, the syntactic component obtains a set of syntactic strings, each representing the topological structure of a picture instead of a user presenting SAP with a syntactic string representing the picture's structure. In other words, the syntactic component of SAP is a translator translating sentences (pictures) of a two-dimensional language L to a one-dimensional language L*.

Given a picture L_i, a structural description L^*_{ij} of L_i is a representation of the primitives and the syntactic relations between the primitives which comprise L_i. This description will be obtained by the syntactic component first as an intermediate tree structure and then as a linear string. The tree structure is actually a pedogogical device to aid in illustrating and conceptional understanding of the syntactic analysis.

To obtain the structural description the syntactic component is composed of a lexicon and two subcomponents, a segmenting subcomponent and a parsing subcomponent. The segmenting subcomponent forms lines from points, higher level primitives such as rectangles and triangles from the lines, and a set of 3-tuples called the triplet set from the higher level primitives (to be called just primitives hereafter) and the syntactic relations between them. The parsing subcomponent tests the set of 3-tuples for completeness, partially orders the triplet set, and then transforms the triplet set first into a tree structure and then into a Reverse Polish string to obtain the final form of the structural description of the pictorial data. The lexicon defines the attributes of the primitives and the syntactic relations which can exist between the primitives. A flow diagram of the syntactic component is shown in Figure 5.

5.2 THE SEGMENTING SUBCOMPONENT

The segmenting subcomponent must first determine and define the lines which compose the pictorial data. The lines are then used to form higher level constituents. To obtain the lines and higher level constituents a grammar K1* is used. K1* is a modified context sensitive phrase structure grammar which is defined as the 4-tuple (T1,N1,R1,SS). The set T1 is the set of terminal symbols of K1*. By the nature of the data, the terminal symbols are discrete points which do not explicitly occur in the grammar. The set N1 is the set of nonterminal symbols. The set R1 is the set of production rules which form higher level constituents from the terminal symbols. The goal of the grammar is SS, the syntactic string.

In addition to the grammar K1*, the segmenting subcomponent uses transformation rules on the line segments formed by K1*. The transformation rules will form additional line segments or concatenate line segments already formed to allow the parsing of the figure to be obtained.

In order to obtain the line segments from the discrete points it is necessary to first label the points. The points are labeled by the directional axes as shown on Page 35.

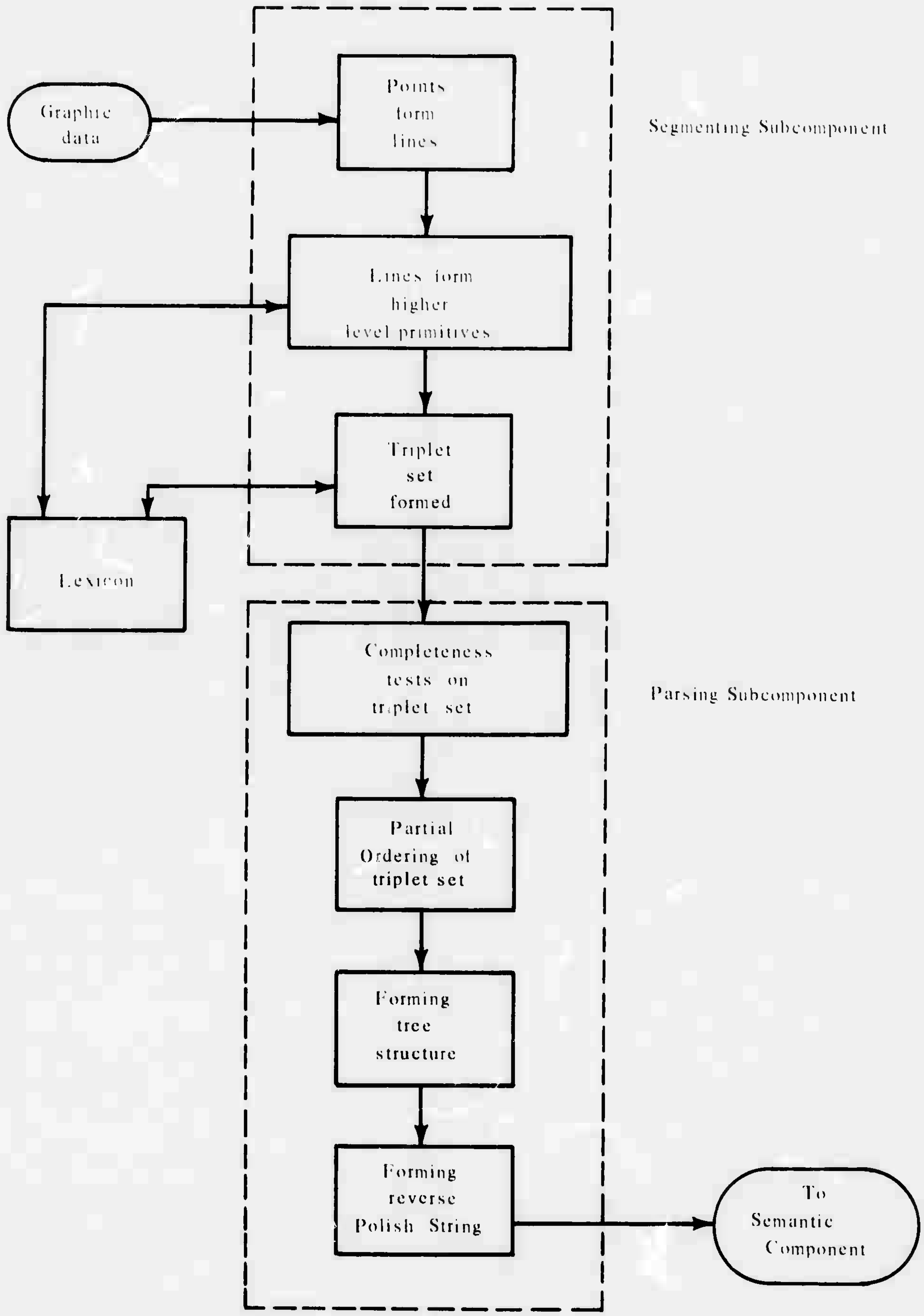
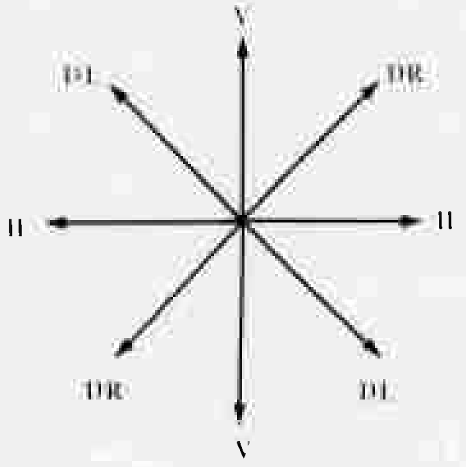


Figure 5. SYNTACTIC COMPONENT



To label the points the axes are moved from point to point, where at each point all neighboring points are labeled according to the axes. Thus, the labeled points indicate their position relative to their neighboring points. Any number of directional axes may be used, differentiating between horizontal, vertical, and various diagonal type lines. Note that a single point may receive multiple labels.

Once the points have been labeled, the line segments are formed from the following productions.

$$
\begin{aligned}
\langle L_H(x,x)\rangle &::= \langle P_H(x)\rangle \\
\langle L_H(y,x)\rangle &::= \langle P_H(y)\rangle\langle L_H(z,x)\rangle \mid \langle L_H(y,z)\rangle\langle P_H(x)\rangle \\
\langle L_V(x,x)\rangle &::= \langle P_V(x)\rangle \\
\langle L_V(y,x)\rangle &::= \langle P_V(y)\rangle\langle L_V(z,x)\rangle \mid \langle L_V(y,z)\rangle\langle P_V(x)\rangle \\
\langle L_{DR}(x,x)\rangle &::= \langle P_{DR}(x)\rangle \\
\langle L_{DR}(y,x)\rangle &::= \langle P_{DR}(y)\rangle\langle L_{DR}(z,x)\rangle \mid \langle L_{DR}(y,z)\rangle\langle P_{DR}(x)\rangle \\
\langle L_{DL}(x,x)\rangle &::= \langle P_{DL}(x)\rangle \\
\langle L_{DL}(y,x)\rangle &::= \langle P_{DL}(y)\rangle\langle L_{DL}(z,x)\rangle \mid \langle L_{DL}(y,z)\rangle\langle P_{DL}(x)\rangle
\end{aligned}
$$

For example, a point x labeled *horizontal*, $P_H(x)$, forms a horizontal line $L_H(x,x)$ whose endpoints are x, i.e., a line composed of one point is formed. To extend this line additional points with the same directional label as the line are concatenated to the already existing line. Concatenating a horizontal point y, $P_H(y)$, to the left side of a horizontal line bounded by points z and x, $L_H(z,x)$, forms a horizontal line bounded by points y and x, $L_H(y,x)$.

When a new line (called an *actual line*) is started because of a change in direction in the points, the last point of the line just generated is called a *vertex point*. Thus, the vertices are end points of lines and indicate juxtaposition points between lines. Vertex points always have multiple labels.

Once the lines of the figure have been determined, the input figure can be defined as a graph composed of a set $V = \{v_i\}$ of points in E^2 and a set of straight lines satisfying the following conditions:

1. Every line contains precisely two points of V, and these agree with its end points.
2. The lines have no common points, except for points of V.

The vertices v_i and v_k which are the end points of a line l are said to be incident with the line. This may be written as l $\mathcal{J}$ (v_i & v_k) which is read as *l joins v_i and v_k*, where the symbol (v_i & v_k) denotes an unordered pair of elements of v.

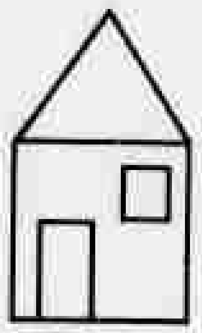
The vertices are placed in a data base where the coordinates of each are stored as attributes of the vertex. This information will be used to determine the length of the line. Another attribute of the vertex v which is continually updated in the data base is its degree $\delta(v)$, which is the number of lines incident with v, a separate count being kept for *actual* lines. Each line is assigned two arguments, the vertices which bound the line. The arguments are defined as follows:

(i) For L_H (v_i,v_j) the first argument is the left hand vertex which bounds the horizontal line.

(ii) For L_V (v_i,v_j) the first argument is the lower vertex which bounds the vertical line.

(iii) For L_{DR} (v_i,v_j) or L_{DL} (v_i,v_j) the first argument is the left hand vertex which bounds the diagonal line.

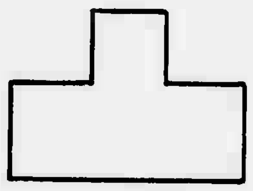
Since the semantic component requires the structural description to be composed of higher level primitives than lines, additional rules must be added to the grammar. This would not necessarily be the case if the pictorial data were one-dimensional figures. For the pictorial data being considered, the higher level primitives are the triangles and rectangles described in the previous chapter. The input figure will be parsed into the higher level primitives by combining the lines determined by K1*. For example, the figure



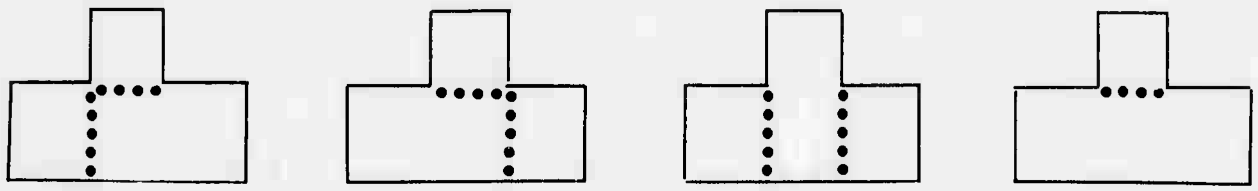
can be readily parsed into three rectangles and a triangle.

However, a difficulty in parsing the pictorial data arises when the higher level primitives required by the semantic component are only implicitly contained in the data. This occurs if, for example, the primitives are two-dimensional and the common boundary between two primitives is removed. In this case the segmenting subcomponent must add lines to the figure to make the primitives explicit. Because there may be a choice as to which lines to add to the figure, more than one configuration of the figure may be possible.

For example, assuming that the structural description is to have rectangles as its primitives, consider the figure on the right:



There are four different configurations for the figure. They are,



where the dotted lines are the added lines. To produce these configurations the following transformation rules are used. These rules are not considered part of the grammar K1* because of the nature of their operation. The brackets on the right hand side indicate a choice in the lines to be added and the numbers on the right hand side of the transformation rule indicates that the corresponding elements on the left hand side of the rule are carried over.

$$L_H(v_i, v_j), L_V(v_j, v_k) \longrightarrow \left\{ \begin{array}{l} (1), (2), L'_H(v_j, v_m) \\ \\ (1), (2), L'_V(v_m, v_j) \end{array} \right\}$$

$$L_H(v_j, v_i), L_V(v_j, v_k) \longrightarrow \left\{ \begin{array}{l} (1), (2), L'_H(v_m, v_j) \\ \\ (1), (2), L'_V(v_m, v_j) \end{array} \right\}$$

The added lines (called *artificial lines*) are designated by a prime so that the parsing subcomponent will be able to indicate in the structural description that the lines were added by the segmenting subcomponent.

Note that when artificial lines are added new vertices may be created. The Appendix 10.2 contains the entire set of rules necessary to add artificial lines to the pictorial data being considered. Each use of an operation rule adds one artificial line to the figure. All four configurations of the sample figure would be operated on by the parsing subcomponent, producing structural descriptions (parsings) to be sent to the semantic component. It is possible that all of the parsings will receive the same semantic interpretation or that some of the parsings cause the figure to be considered anomalous, due to limitations in the semantic component, and hence assigned no semantic interpretation.

Once the adding of artificial lines to the pictorial data is complete, it is necessary to explicitly define the higher level primitives which comprise the data. This is done by combining the lines into the higher level constituents such as triangles and rectangles by the following steps.

(1) Artificial lines which were added by the segmenting subcomponent are concatenated with actual lines of the figure. Two lines are considered for concatenation if an artificial line has the same direction as an actual line and they have a common end point.

(2) The artificial and actual lines are concatenated if their common vertex v has a degree, $\delta(v)=3$.

(3) If $\delta(v)=4$, the artificial and actual lines cannot be concatenated.

If two lines are concatenated, the two lines before the concatenation are not lost since they may also be needed in the formation of higher level constituents. The concatenated line is primed. This may be written as shown below where, for the first rule, the new line formed is $L'_1(v_i,v_k)$.

$$L_1(v_i,v_j).L_1(v_j,v_k).L_m(v_j,v_n) \longrightarrow L'_1(v_i,v_k).(3)$$
$$L_1(v_i,v_j).L_1(v_j,v_k).L_m(v_n,v_j) \longrightarrow L'_1(v_i,v_k).(3)$$

Once all possible line concatenations are made, a grammar will form the higher level primitives required by the semantic component. The grammar rules which form these constituents are of the form,

$$<X(w,h,v)> ::= <L^{(')}_m(v_i,v_j)><L^{(')}_n(v_k,v_l)>\ldots/\emptyset$$

The rule states that constituent of type X may be formed from the juxtaposed lines in the context of restriction $\emptyset$. The arguments of constituent X are defined as:

(i) w is a number assigned to each constituent identifying that constituent from other constituents of the same type;

(ii) h is the horizontal dimension of the constituent:

(iii) v is the vertical dimension of the constituent.

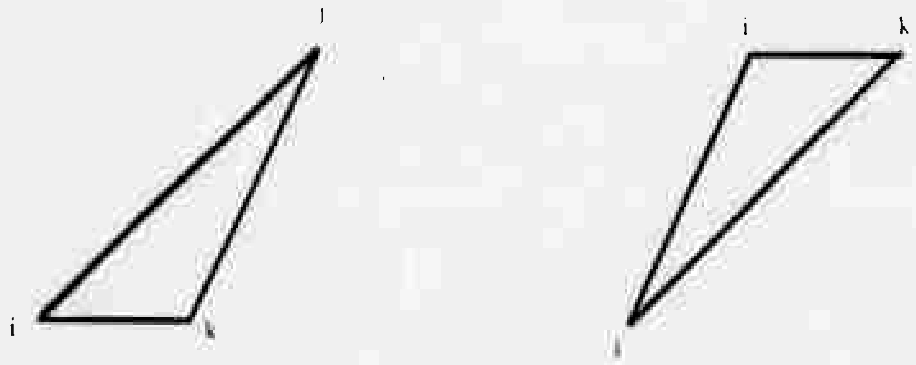
The primes enclosed in parentheses indicate they are optional and thus a line may be artificial, actual, or a concatenation of the two. The restriction $\emptyset$ refers to an entry in the lexicon which contains additional restrictions for forming the constituent and defines the attributes of the constituent being formed. The lexicon entry indicates how the horizontal and vertical dimensions of the constituent are to be determined and other attributes which will be required to complete the structural description of the figure. When the constituent is formed, its entry is made in the data base and the vertices which are the arguments of the lines forming the constituent are listed as an attribute of the constituent.

Examples from the set of production rules to form the higher level primitive constituents required for the pictorial data being considered are shown in Figure 6. The listing of the grammar K1* in Appendix 10.2 contains the complete set of rules for all eight primitives indicated in Chapter 4.

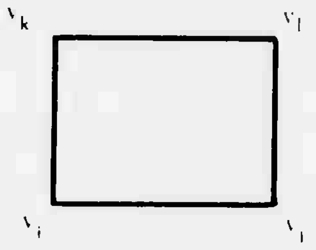
In contrast to juxtaposed elements in linguistics, the above rules do not require the juxtaposed elements to be ordered. The arguments of the lines are ordered and thus eliminate any possible ambiguity between figures. For example, assume the following rule where the arguments are not considered ordered (TS stands for scalene triangle)

$$<TS(w,h,v)> ::= <L_{DR}(v_i,v_j)><L_{DR}(v_k,v_j)><L_H(v_i,v_k)>$$

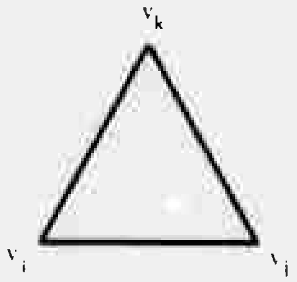
so either of the two figures may be produced.



$\langle R(w,h,v)\rangle ::= \langle L_H(v_i,v_j)\rangle\langle L_V(v_i,v_k)\rangle\langle L_H(v_k,v_l)\rangle$
$\langle L_V(v_j,v_l)\rangle$ / RECTANGLE



$\langle TI(w,h,v)\rangle ::= \langle L_H(v_i,v_j)\rangle\langle L_{DR}(v_i,v_k)\rangle$
$\langle L_{DL}(v_k,v_j)\rangle$ / ISOSCELES TRIANGLE



$\langle C(w,h,v)\rangle ::= \langle L_H(v_i,v_j)\rangle\langle L_{DL}(v_j,v_k)\rangle\langle L_V(v_l,v_k)\rangle$
$\langle L_{DR}(v_m,v_l)\rangle\langle L_H(v_n,v_m)\rangle\langle L_{DL}(v_p,v_n)\rangle$
$\langle L_V(v_p,v_r)\rangle\langle L_{DR}(v_r,v_i)\rangle$ / CIRCLE

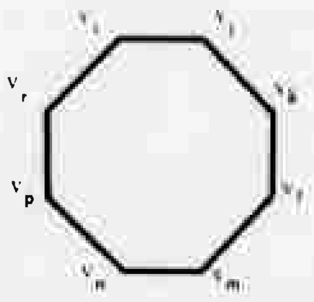


Figure 6. Examples of Production Rules to Form Higher Level Primitives

But if the arguments of the lines are considered ordered and defined as earlier, only the left hand figure is possible.

Having formed all the higher level primitives from the actual and artificial lines, in order to provide a structural description of the pictorial data, a set of 3-tuples are created, composed of pairs of contiguous primitives and their respective syntactic relation. The primitives are paired by finding a subset of the vertices of one primitive to also be a subset of the vertices of a second primitive. The correct syntactic relation is determined by their definitions in the lexicon.

The general form of a lexicon entry for either a primitive or a syntactic relation is of the following form.

NAME: form

Requirements: $F(l_m)\ \Psi_i\ G(l_t)$

Assignments: $m_i = \begin{cases} K(l_m)\ 0_i W(l_t) \\ n_k \end{cases}$

The entry NAME can be either the name of a primitive or a syntactic relation. The portion labeled Requirements lists any restrictions which SAP must consider. The functions F and G represent general functions such as length of a line segment and Ψ_i is an arithmetic relation or a relation such as contiguous, "$\boxplus$". The portion labeled Assignments assigns to variable m_i either (i) a numeric value which results from the calculation of the indicated expression where K and W are functions which obtain the x and y coordinates of the arguments l_m and l_t which are line segments or vertices, or (ii) a label n_k which indicates the directional value of an argument of a syntactic relation. The lexicon entries for the eight primitives and the three basic syntactic relations are listed in Appendix 9.2. An example of the entry for the relation *on top of* is shown below.

ON TOP OF: $X, Y, {*}R1(n_1, n_2 \cap n_3 \cap n_4)$

Requirement: $X(BOT) \boxplus Y(TOP)$

Assignments: $n_1 = 0$

$n_2 = LE$

if $XCOORD(X_{LE}) \geq XCOORD(Y_{LE})$

then,

$n_3 = XCOORD(X_{LE}) - XCOORD(Y_{LE})$

$n_4 = RT$

if $XCOORD(X_{LE}) < XCOORD(Y_{LE})$

then,

$n_3 = XCOORD(Y_{LE}) - XCOORD(X_{LE})$

$n_4 = LT$

The 3-tuples constitute a set of triplets, $T = \{t_i\}$. Thus triplets are of the form, $t_i = (p_k, p_l, s_m)$ where p_k and p_l are members of the set P of primitives and s_m is a member of the set S of syntactic relations, and where p_k and p_l are contiguous primitives, and s_m is the syntactic relation between them. The definitions of the syntactic relations may be found in Chapter 4. While the syntactic language in Chapter 4 defined six syntactic relations, only three of them are actually necessary,

*R1 = *on top of*

*R3 = *to the left of*

*R5 = *is contained within*

the remaining three being just the inverse of the preceding three. To normalize the graphic data and reduce the number of different structural descriptions which must be considered, only the above three syntactic relations are used internally by SAP. If a user provides SAP with a syntactic string using relations other than the three indicated, the relations are converted to the three above relations by easily manipulating the syntactic string.

Triplets may also be of the form (X,0,0). This occurs if the primitive X occurs in the figure but is not contiguous to any other primitive in the figure. Triplets of this and other forms require various tests and transformations to be performed on the triplet set. Once all possible triplets have been formed, the triplet set T is sent to the parsing subcomponents where it is checked for completeness and then transformed into a structural description of the figure.

5.3 THE PARSING SUBCOMPONENT

5.3.1 COMPLETENESS TESTS

Once the triplet set has been formed from all pairs of contiguous primitives and the syntactic relations between them, the triplet set is tested by the parsing subcomponent for completeness. To facilitate explaining the completeness tests the triplet set will be represented by a directed graph. Each edge of the graph has associated with it an orientation resulting from the end points of each edge which constitute an ordered pair of vertices.

The directed graph D is a triple (P,S,Λ) consisting of a nonempty set P, a set S which is disjoint from P, and a mapping Λ of S into P X P. P X P is the cartesian product of a set P with itself forming all ordered pairs (p_i,p_j) such that $p_i \in P$ and $p_j \in P$.

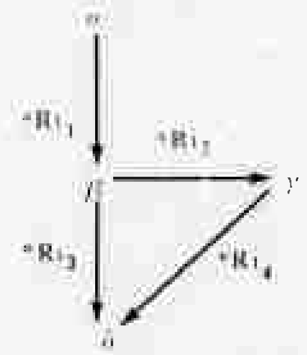
(i) The elements of P are called vertices and represent the primitives which are contained in the triplets.

(ii) The elements of S are called directed edges and represent the syntactic relations which are contained in the triplets.

(iii) Λ is called the direct incidence mapping associated with D.

If $s \in S$ and $\Delta(s) = (p_i,p_j)$ then the directed edge is said to have p_i as its initial vertex and p_j as its terminal vertex or $s \simeq (p_i,p_j)$.

Thus, the directed graph is isomorphic to the triplet set. For example, the triplet set $(\alpha,\beta,*Ri_1)$, $(\beta,\gamma,*Ri_2)$, $(\beta,\delta,*Ri_3)$, $(\gamma,\delta,*Ri_4)$ would form the directed graph,


$$\begin{tikzcd}
\alpha \arrow[d, "*Ri_1"'] & \\
\beta \arrow[r, "*Ri_2"] \arrow[d, "*Ri_3"'] & \gamma \arrow[dl, "*Ri_4"] \\
\delta &
\end{tikzcd}$$

It is interesting to note that the directed graph representing the triplet set could be obtained directly from the two-dimensional picture serving as input. As indicated in Chapter Two, the figure itself may be considered a geometric graph. A modified dual of this geometric graph is the desired directed graph representing the triplet set. To obtain the modified dual:

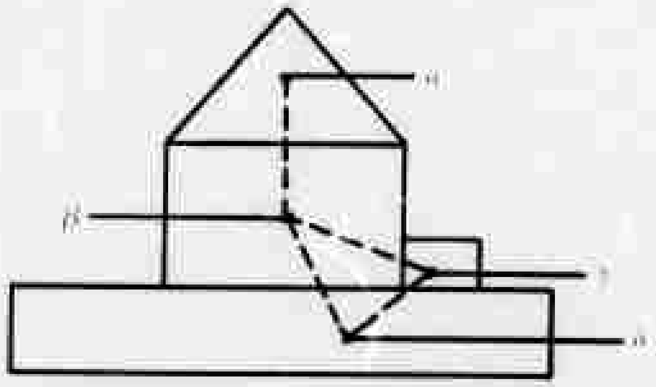
(i) Consider the geometric graph (figure) with regions $R_i (i=1,\ldots,n)$. Associate a point p_i with each region R_i by choosing one of the points within the region. If two regions R_i and R_j are adjacent, join p_i and p_j by an edge $p_i p_j$ which intersects the common boundary of R_i and R_j only once and has no point in common with any other boundary of the graph.

(ii) No point p_i is assigned to the region surrounding the graph.

(iii) The edge $p_i p_j$ receives a direction and label which corresponds to the *syntactic relation* ordering between regions R_i and R_j.

This procedure yields a new graph D′ with vertices $p_1,\ldots,p_n$. It is called a modified dual graph of D.

For example, consider the picture below.



The point p_i of region R_i is indicated by the dot and labeled by the greek letter shown. Connecting the dots by the indicated dotted lines and giving a direction and label to the corresponding connections provides the following the directed graph.

$$\begin{tikzcd}
\alpha \arrow[d, "*Ri_1"'] & \\
\beta \arrow[r, "*Ri_2"] \arrow[d, "*Ri_3"'] & \gamma \arrow[dl, "*Ri_4"] \\
\delta &
\end{tikzcd}$$

While the completeness tests are presented in terms of the directed graph to provide ease of explanation, the tests are also described in terms of the transformation rules which actually operate on the triplet set. The general transformation rule is of the form, $T1 \longrightarrow T2$, where T1 and T2 are strings of triplets of the form $(\alpha_1,\beta_1,{*Rj_1}),(\alpha_1,\beta_2,{*Rj_2}),\ldots(\alpha_n,\beta_n,{*Rj_n})$. T1 is a subset of the original triplet set T and T2 is the subset T1 after it has been transformed. As indicated in the previous section, triplets may also be of the form $(\alpha,0,0)$ where β_i and $*Rj_i$ are zero. As a shorthand notation, T2 may have members of its string of the form (n) which indicates that the nth triplet on the left hand side of the rule, T1, should be transferred to the right hand side, T2, unmodified. Triplets may be removed from, added to, or modified in the triplet set. The order of the triplets in T1 and T2 is arbitrary, though it will be shown in a later section that it is necessary to partially order the triplet set. The arguments of the primitives and syntactic relations which were described in the last chapter are not included in the examples of this chapter as a matter of convenience.

There are three completeness tests which are performed on the triplet set. These tests are,

1. The test for *isolated primitives.* An isolated primitive occurs when a primitive is contained within another primitive in the figure and the former primitive is not contiguous to any primitives.
2. The test for *missing relations.* A missing relation occurs when two primitives,which are not contiguous, are both syntactically related to a third primitive by the same type of relation.
3. The test for *inconsistent relations.* An inconsistent relation occurs when two contiguous primitives are syntactically related when they should not be.

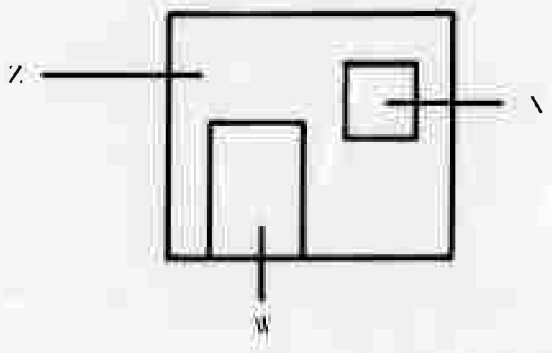
The completeness tests will now be described in detail.

1. The test for *isolated primitives* is essentially a check to determine whether the undirected graph G which corresponds to the directed graph D is connected. The test for connectedness of the undirected graph may be defined as follows:

> A finite sequence $s_1, s_2, \ldots, s_n$ of edges of the corresponding undirected graph constitutes an *edge progression* of length N if there exists an appropriate sequence of n + 1 vertices $p_0, p_1, \ldots p_n$ such that s_i $(p_{i-1} \& p_i)$ for i = 1,2,...,n. The set of edges, without regard to sequencing, is said to constitute a *chain.* Finally, a graph is said to be *connected* if every pair of distinct vertices are joined by at least one chain. Other graphs are said to be disconnected.[41]

The graph G is not connected if its edges can be partitioned into two subsets S_1 and S_2 such that both end points of every edge are in the same subset. There are two cases of isolated primitives to be considered, depending on whether P_1 and or P_2, the subsets resulting from the partitioning of the set of vertices, consists of an isolated vertex p_i that is, $\delta(p_i)$ 0.

(i). The first case of an isolated primitive arises when P_1 or P_2 consists of a single vertex. Hence, either S_1 or S_2 is null. This may occur in a figure when a primitive X is contained within a primitive Z, but primitive X is not contiguous to primitive Z or any other primitives which may be within Z. For example,



The directed graph for this example is

$$\begin{tikzcd} Z \arrow[d, "*R5"] & & X \\ W & & \end{tikzcd}$$

The undirected graph is,

$$\begin{tikzcd} Z \arrow[d, dash, "*R5"] & & X \\ X & & \end{tikzcd}$$

To form a connected graph the vertex X must be connected by an edge, representing a syntactic relation, to either Z or W. Since in the general case there may be several primitives contained within Z to which X can be syntactically related, X will be related to Z, the terminal point of the *contained within* relation. The resulting undirected and directed graphs are,

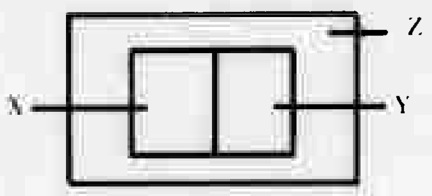
$$\begin{tikzcd} Z \arrow[r, dash, "*R5"] \arrow[d, dash, "*R5"'] & X \\ W & \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} Z \arrow[r, "*R5"] \arrow[d, "*R5"] & X \\ W & \end{tikzcd}$$

The triplet from which the isolated primitive results is of the form $(\gamma,0,0)$. The triplet set for the example is (W,Z,*R5), (X,0,0) and it is transformed to (W,Z,*R5),(X,Z,*R5). The transformation rule for this type of isolated primitives is

$$(\alpha,\beta,{*}R5),\ (\gamma,0,0) \longrightarrow (1),\ (\gamma,\beta,{*}R5)$$

The triplets of the left are members of the original triplet set. The triplets on the right are the result of a transformation on the original triplets, the (1) indicating that the first triplet remains the same while the second triplet is transformed as indicated.

The first case of an isolated primitive with P_1 or P_2 consisting of a single vertex may also occur when a primitive X is contained within a primitive Z, and while X is contiguous to at least one primitive Y which is contained within Z, there are no primitives contiguous to Z. For example,



The corresponding directed and undirected graphs are,

$$\begin{tikzcd} Z & \\ X \arrow[r, "*R2"'] & Y \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} Z & \\ X \arrow[r, dash, "*R2"'] & Y \end{tikzcd}$$

To form a connected graph the vertex Z is connected by an edge to one of the primitives contained within Z, the choice being arbitrary. The resulting undirected and directed graphs are, choosing X for instance,

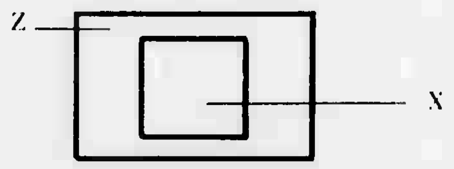
$$\begin{tikzcd} Z \arrow[d, dash, "*R5"] & & & Z \\ X \arrow[r, dash, "*R2"'] & Y & \text{and} & X \arrow[u, "*R5"'] \arrow[r, "*R2"'] & Y \end{tikzcd}$$

The triplet from which the isolated primitive results is again of the form, $(\gamma,0,0)$. The triplet set for the example is (Z,0,0) (X,Y,*R2) and it is transformed to (X,Z,*R5),(X,Y,*R2). The transformation rule for this type of isolated primitive is,

$$(\alpha,\beta,*Ri),\ (\gamma,0,0) \longrightarrow (1),\ (\alpha,\gamma,*R5)$$

In the actual implementation, the choice as to which of the two transformation to use is made by checking the coordinates of the vertices of the primitives. The vertices are attributes of the primitives though they have not been indicated here.

The occurrence where both P_1 and P_2 consist of a single vertex can be easily handled. This arises when primitive X is contained within primitive Z, and both X and Z are contiguous to no primitives. For example,



The directed and undirected graphs are,

$$\begin{tikzcd} Z & X \end{tikzcd}$$

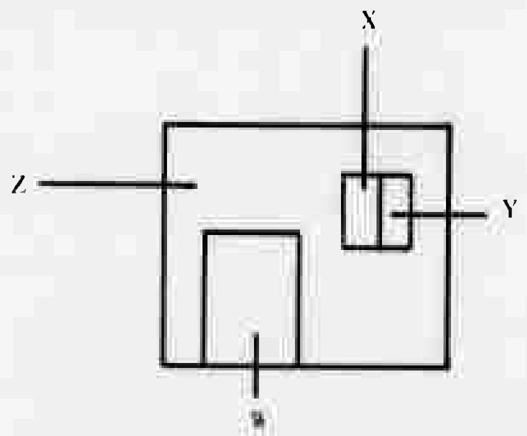
which may be connected to form,

$$\begin{tikzcd} Z \arrow[r, "*R5"'] & X & \text{and,} & Z \arrow[r, dash, "*R5"'] & X \end{tikzcd}$$

The triplet set for the example is of the form (Z,0,0) (X,0,0) and is transformed to (X,Z,*R5). The transformation rule for this type of isolated primitive is,

$$(\alpha,0,0),\ (\beta,0,0) \longrightarrow (\beta,\alpha,*R5)$$

(ii). The second case of an isolated primitive arises when the partitioning of the unconnected graph G provides two subsets P_1 and P_2, both of which contain more than one vertex. This occurs in a figure when a primitive X is contiguous to at least one primitive Y, and while both X and Y are contained within a primitive Z, neither X nor Y is contiguous to Z or any other primitives which are contained within Z. For example,



The directed is

$$\begin{tikzcd} Z & & \\ W \arrow[u, "{*R5}"'] & X \arrow[r, "{*R2}"] & Y \end{tikzcd}$$

to which is added a relation between Z and either of the primitives not connected to Z.

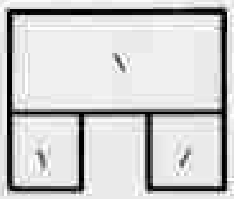
The triplet set for the example is of the form (W,Z,*R5), (X,Y,*R2) and is transformed to (W,Z,*R5), (X,Y,*R2) (X,Z,*R5). The transformation rule is,

$$(\alpha,\beta,*R5), (\gamma,\delta,*Ri) \longrightarrow (1), (2), (\gamma,\beta,*R5)$$

As with the previous transformation rules, their application is in part determined by checking the attributes of the primitives.

2. The test for *missing relations* is to determine when two noncontiguous primitives should be related. The addition of these relations allows further parsings of the figure to be performed, the adding of relations being the inclusion of more triplets in the triplet set.

An example of a figure which results in a missing relation is,



The triplet set obtainable from the contiguous primitives is,

(X,Y,*R1), (X,Z,*R1)

The directed graph formed from these triplets is,

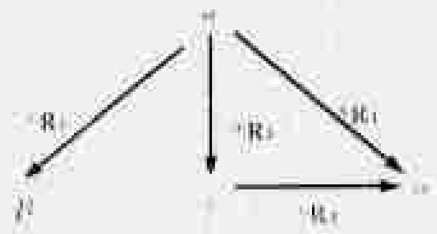
$$\begin{tikzcd} & X \arrow[dl, "{*R1}"'] \arrow[dr, "{*R1}"] & \\ Y & & Z \end{tikzcd}$$

It will be shown in the next section that two parsings may be obtained from the triplet set. A third relation can be represented between Y and Z. The triplet representing this relation is (Y,Z,*R2) and by the addition of this triplet to the triplet set four more parsings of the figure can be obtained. After the third relation is added the directed graph is,

$$\begin{tikzcd} & X \arrow[dl, "{*R1}"'] \arrow[dr, "{*R1}"] & \\ Y \arrow[rr, "{*R2}"'] & & Z \end{tikzcd}$$

The missing relation test searches the directed graph for the presence of two or more directed edges which have the same syntactic label and the same initial or terminal vertex. A relation is considered missing if the noncommon vertices of the directed edges just described are not connected by a path which does not contain the common vertex. There are two types of cases in which a missing relation may arise, depending on whether the common vertex is an initial or terminal vertex.

If the directed edges with the same syntactic label have the same initial vertex,



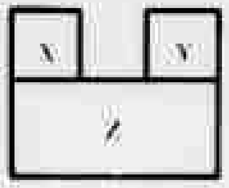
the missing relation is between β and γ where *Ri cannot be a *contained within* relation. It will be pointed out in the discussion on the inconsistency test that *Rj also cannot be a *contained within* relation, and of course i ≠ j. The transformation rule which operates on the triplet set for the situation of the directed edges having the same initial vertex is,

$$(\alpha,\beta,{*}Ri),\ (\alpha,\gamma,{*}Ri) \longrightarrow (1),\ (2),\ (\beta,\gamma,{*}Rj)$$

The directed graph for the directed edges having the same terminal node is,

$$\begin{tikzcd} \alpha \arrow[dr, "{*}Ri"'] & & \beta \arrow[dl, "{*}Ri"] \\ & \gamma & \end{tikzcd}$$

where again there is a missing relation between α and β. An example of a figure with such a missing relation is,

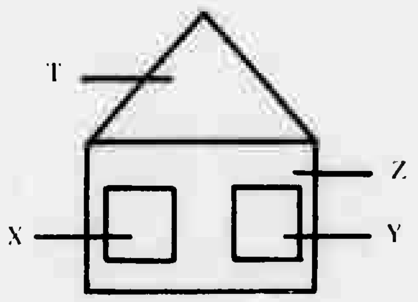


which has the directed graph,

$$\begin{tikzcd} X \arrow[dr, "{*}R1"'] & & Y \arrow[dl, "{*}R1"] \\ & Z & \end{tikzcd}$$

As pointed out earlier, by adding a triplet which relates X and Y to the triplet set will allow additional parsings of the figure if *R1 is not a *contained within* relation.

However, if *R1 is a *contained within* relation it is mandatory to determine the missing relation between X and Y. Because primitives related to Z by a *contained within* relation are are not related in the triplet set, it will not be possible to parse the figure. The absence of the syntactic relation between X and Y from the triplet set may be because either X or Y was an isolated primitive which was placed in a triplet to remove a case of incompleteness. For example,



The graph for the preceding figure is,

$$\begin{tikzcd}
X \arrow[dr, "{*}R5"'] & & Y \arrow[dl, "{*}R5"] \\
& Z \arrow[d, "{*}R1"] & \\
& T &
\end{tikzcd}$$

Syntactically relating X and Y results in the graph,

$$\begin{tikzcd}
X \arrow[rr, "{*}R2"] \arrow[dr, "{*}R5"'] & & Y \arrow[dl, "{*}R5"] \\
& Z & \\
& T \arrow[u, "{*}R1"'] &
\end{tikzcd}$$

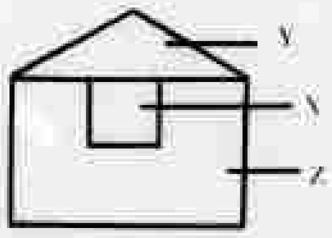
The transformation rule which operates on the triplet set for the situation of the directed edges having the same terminal vertex is,

$(\alpha,\beta,{*}Ri), (\gamma,\beta,{*}Ri) \longrightarrow (1), (2), (\alpha,\gamma,{*}Rj)$, where $i \neq j$.

The problem of obtaining all possible syntactic relations which can occur between primitives contained within a given primitive is not solved by this formulation. Though it is only a minor extension of the above, and the additional triplets will provide more parsings, it is not clear at this point that all such possible parsings are necessary or even desired.

3. The test for what may be considered *inconsistent relationships* must also be applied to the triplet set before any attempt is made to parse the figure. The relations are really not inconsistent but would appear so to the parsing algorithm when an attempt is made to use them and so they must be removed from the triplet set.

(i) The first case of an inconsistent relationship between a primitive X and a primitive Y can be determined on the directed graph of the figure by a vertex X which is, on the corresponding undirected graph, adjacent (α and β are called adjacent vertices if $s \sim (\alpha\&\beta)$ for at least one edge) to a vertex Y where X is also the initial vertex on the directed graph of a directed edge labeled by a *contained within* relation and which has Z as the terminal vertex and Y is also adjacent to Z by a directed edge which has the same label as the directed edge connecting X and Y. For example,



has the directed graph,

$$\begin{tikzcd}
 & Y \arrow[dl, "*R1"'] \arrow[dr, "*R1"] & \\
X \arrow[rr, "*R5"'] & & Z
\end{tikzcd}$$

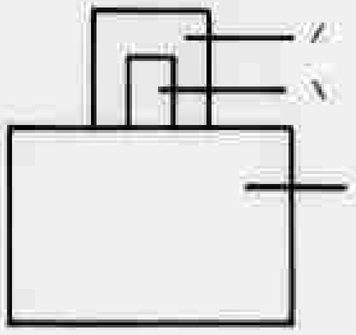
The resulting directed graph is,

$$\begin{tikzcd}
 & Y \arrow[d, "*R1"] \\
X \arrow[r, "*R5"] & Z
\end{tikzcd}$$

The transformation rule which removes the triplet from the triplet set is,

$$(\overline{\alpha,\beta},*Ri),\ (\beta,\gamma,*R5),\ (\overline{\alpha,\gamma},*Ri) \longrightarrow (2),\ (3)$$

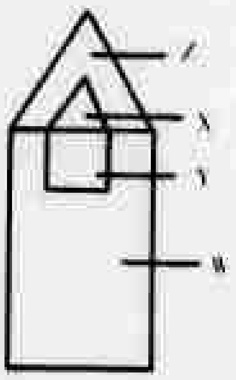
Note that the bar over the triplets indicates that the order of the primitives can be reversed. However, reversing the order in one of the triplets of a rule necessitates reversing the order in all of the triplets of that rule which have a bar over them. The reverse ordering of the triplets will eliminate inconsistencies which occur in a figure such as,



using the transformation rule,

$$(\beta,\alpha,*Ri),\ (\beta,\gamma,*R5),\ (\gamma,\alpha,*Ri) \longrightarrow (2),\ (3)$$

(ii) The second case of an inconsistent relation between X and Y is determined on the directed graph by a vertex X which is adjacent to a vertex Y and where X is an initial node of a directed edge to a vertex Z which is labeled by a *contained within* relation and Y is an initial node of a directed edge to a vertex W which is labeled by a *contained within* relation and Z and W are not the same vertex. For example,



which has the directed graph,

$$\begin{tikzcd}
 & Z \arrow[ddl, "*R1"'] \arrow[ddr, "*R1"] & \\
 & X \arrow[u, "*R5"] \arrow[dl, "*R1"] \arrow[dr, "*R1"'] & \\
Y \arrow[rr, "*R5"'] & & W
\end{tikzcd}$$

Note that the first rule for inconsistent relations will reduce the directed graph to,

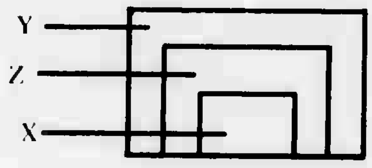
$$\begin{tikzcd}
 & Z \arrow[ddr, "*R1"] & \\
 & X \arrow[u, "*R5"] \arrow[dl, "*R1"] & \\
Y \arrow[rr, "*R5"'] & & W
\end{tikzcd}$$

The second transformation rule for inconsistent relations removes the relation between X and Y. This rule is,

$$(\alpha,\beta,{*}R5),\ (\alpha,\gamma,{*}Ri),\ (\beta,\delta,{*}Ri),\ (\gamma,\delta,{*}R5) \longrightarrow (1),\ (3),\ (4)$$

In the example, $\alpha = X$, $\beta = Z$, $\gamma = Y$ and $\delta = W$.

(iii) The third case of an inconsistent relation between X and Y is determined on the directed graph by a vertex X, which is the initial vertex for a directed edge to vertex Z which is labeled by a *contained within* relation and X is also the initial vertex for a directed edge to vertex Y which is labeled by a *contained within* relation and Z is the initial vertex for a directed edge to Y which is labeled by a *contained within* relation. For example, the figure



has the directed graph

$$\begin{tikzcd}
X \arrow[rr, "*R5"] \arrow[dr, "*R5"'] & & Z \arrow[dl, "*R5"] \\
 & Y &
\end{tikzcd}$$

The resulting directed graph is,

$$\begin{tikzcd} X \arrow[dr, "*R5"'] & & Z \arrow[dl, "*R5"] \\ & Y & \end{tikzcd}$$

and the transformation rule which removes the triplet from the triplet set has the same form as the rule before last, where i = 5.

$$(\beta,\alpha,*R5),\ (\beta,\gamma,*R5),\ (\gamma,\alpha,*R5) \longrightarrow (2),\ (3)$$

The transformation rules are summarized in Appendix 10.2. They have been created to process the triplet set, the final triplet set then being transformed into a tree structure and Reverse Polish string. The transformation rules have developed out of the nature of the pictorial data being considered and the primitives and syntactic relation used in the grammar to structurally describe the data. The rules are not expected to beexhaustive though they will serve a great many configurations of the primitives and syntactic relations defined in the previous chapter. Once the triplet set is considered complete the set is partially ordered.

5.3.2 PARTIAL ORDERING OF THE TRIPLET SET

Following the testing of the triplet set for completeness and before any tree structure can be formed from the triplet set, the triplet set must be partially ordered. The partial ordering forms subsets of the triple set, which in turn allows the tree structures or parsings to be readily obtained from the triplet set. The partial ordering is as follows

(1) The triplet set is scanned for a triplet which contains a *contained within* relation. The initial and terminal vertices of this triplet are subscripted throughout the triplet set with the number, k, of the subset being formed. The triplet containing the *contained within* relation is then removed from the triplet set, but must be replaced in the triplet set if Step 3 is not found applicable to any triplets.

(2) The triplet set is now scanned for other triplets which have the same terminal vertex as the terminal vertex in Step 1 and also a *contained within* relation. The initial vertex of any such triplet is subscripted throughout the triplet set with the number k. The triplet found in Step 2 with the *contained within* relation is then removed from the triplet set.

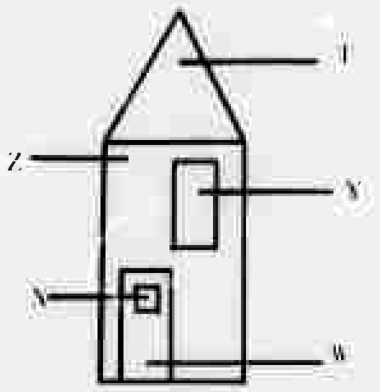
(3) The triplet set is now scanned for triplets which have vertices which have a k subscript except those which have a terminal vertex which is the same as the terminal vertex of the triplet found in Step 1. These triplets are placed in the k^{th} subset. The vertices of any triplet placed in the k^{th} subset are subscripted throughout the triplet set with a k subscript.

If a triplet which is k subscripted contains a *contained within* relation, the vertices of the triplet are desubscripted throughout the triplet set, and Step 1 is repeated where k is set to k+1. However, before the k+1 subset is formed, the k subset is completed.

(4) When no more triplets can be added to the k^{th} subset, the k+1 subset is formed. If there is no k+1 subset to be formed, Step 1 is repeated. If Step 1 produces no new triplets to be subscripted, the unsubscripted vertices of the triplet set are subscripted with a 0.

An example of the partial ordering of a triplet set is shown in Figure 7. Once the triplet set has been partially ordered into subsets, a partial ordering is placed on the triplets in the subsets. The partial ordering within the subsets can be described by the following:

(5) If three triplets in a subset satisfy either of the two rules listed below then the three



(T,Z,*R1), (W,Y,*R2), (W,Z,*R5), (X,W,*R5), (Y,Z,*R5)

Step 1. $(T,Z,*R1), (W,Y,*R2), (W_1,Z_1,*R5), (X,W,*R5), (Y,Z,*R5)$

$(T,Z_1,*R1), (W_1,Y,*R2), (X,W_1,*R5), (Y,Z_1,*R5)$

Step 2. $(T,Z_1,*R1), (W_1,Y_1,*R2), (X,W_1,*R5), (Y_1,Z_1,*R5)$

$(T,Z_1,*R1), (W_1,Y_1,*R2), (X,W_1,*R5)$

Step 3. $(T,Z_1,*R1), \underbrace{(W_1,Y_1,*R2)}_{\text{Subset 1}}, (X,W_1,*R5)$

Step 1. $(T,Z_1,*R1), \underbrace{(W_2,Y_1,*R2)}_{\text{Subset 1}}, \underbrace{(X_2,W_2,*R5)}_{\text{Subset 2}}$

Step 2,3. $\emptyset$

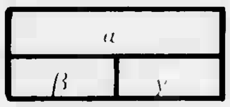
Step 4. $\underbrace{(T_0,Z_1,*R1)}_{\text{Subset 0}}, \underbrace{(W_2 Y_1,*R2)}_{\text{Subset 1}}, \underbrace{(X_2,W_2,*R5)}_{\text{Subset 2}}$

Figure 7. Example of Partial Ordering of a Triplet Set

triplets are subscripted by the number of the mth such grouping in the kth subset. Note that the triplet can be subscripted more than once by these rules, where $i,j \neq 5$.

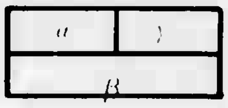
(i) $(\alpha,\beta,{*}Ri)$, $(\alpha,\gamma,{*}Ri)$, $(\beta,\gamma,{*}Rj)$

For example,



(ii) $(\alpha,\beta,{*}Ri)$, $(\gamma,\beta,{*}Ri)$, $(\alpha,\gamma,{*}Rj)$

For example,

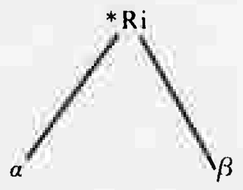


The partial ordering of the triplet set into subsets is to enable sublevels of syntactic tree structures to be formed easily. The ordering within the subsets is to allow particular triplets to be suppressed while a tree structure is being formed. Because the triplets are eventually to be used, they are suppressed but not removed from the triplet set.

5.3.3 THE SYNTAX TREE

Once the triplet set has been partially ordered it is transformed into a set of binary tree structures. Each tree structure represents a parsing of the figure where only primitives are allowed to occur as terminal points (leaves) of the tree and only syntactic relations to occur at nonterminal points. As indicated in 5.1, the following discussion on the tree structure is presented only to illustrate the formation of the structural description. Section 5.3.4 describes the process by which the structural description in the form of a string can be obtained directly from the triplet set.

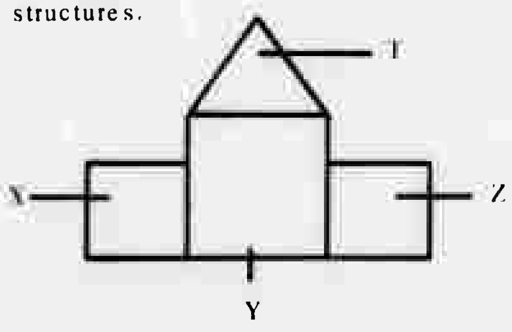
The tree structure is formed by a chaining process which operates on linked triplets. Two triplets are considered linked if they have a common primitive. In order to form a tree an arbitrary triplet is chosen as the starting point and is written as a binary tree with two leaves. Thus, the triplet $(\alpha,\beta,{*}Ri)$ forms the tree,



where the tree structure preserves the ordering of the elements within the individual triplets.

The chaining process now replaces the leaves of the tree with the syntactic relations and corresponding primitives taken from the triplets which are linked to the triplets which compose the tree. This process is continued until the triplet set is exhausted.

For example, the figure and corresponding triplet set transform into the following tree structures.



gives

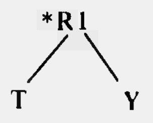
$$\begin{tikzcd} & T \arrow[d, "{*}R1"] & \\ X \arrow[r, "{*}R2"'] & Y \arrow[r, "{*}R2"'] & Z \end{tikzcd}$$

or,

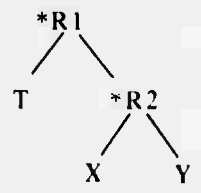
(T,Y,*R1)
(X,Y,*R2)
(Y,Z,*R2)

There is no need to subscript the primitives of the triplet set since no subsets will be formed. The steps in building the tree structure are listed, where the (T,Y,*R1) triplet is chosen first in building the first tree structure.

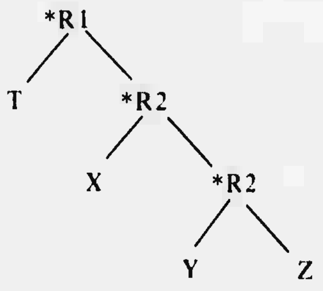
1. (i)



(ii)

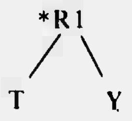


(iii)

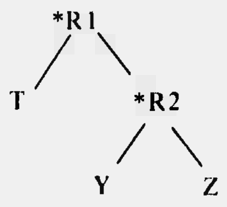


For completeness, the other five parsings are shown.

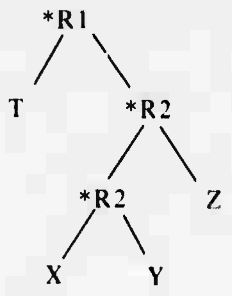
2. (i)



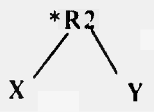
(ii)



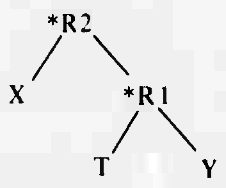
(iii)



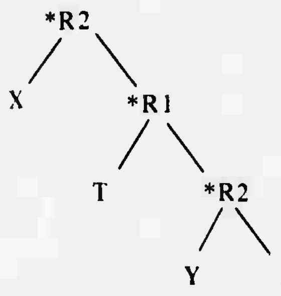
3. (i)



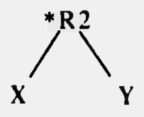
(ii)



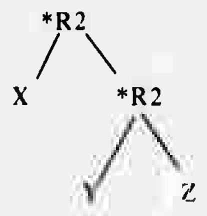
(iii)



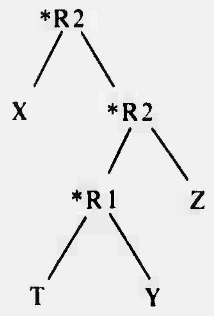
4. (i)



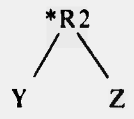
(ii)



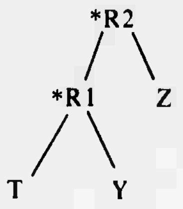
(iii)



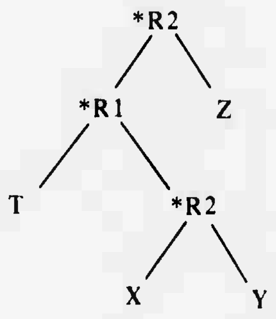
5. (i)



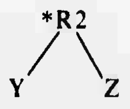
(ii)



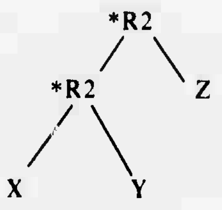
(iii)



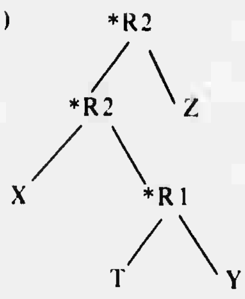
6. (i)



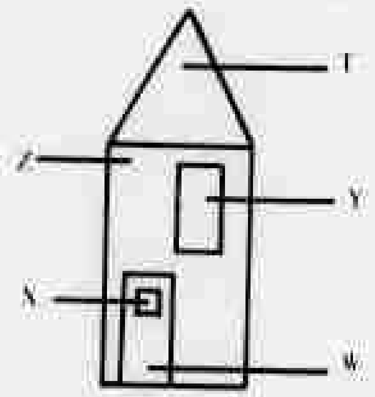
(ii)

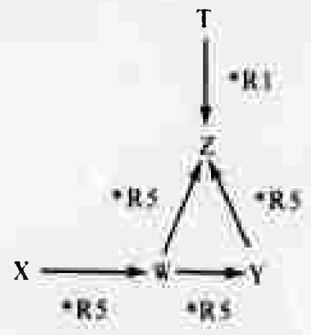


(iii)



The tree structures presented in the example may be considered as existing on a single level. However, the portion of the tree representing the syntactic relations of a primitive which is the initial vertex (on the directed graph representation of the triplet set) of a *contained within* relation or is syntactically related to a primitive which is the initial vertex of a *contained within* relation, drops to a secondary level. A double line represents the *contained within* relation and annexes the secondary level in the tree structure. The nth level of the tree structure can have a substructure annexed to it, forming an n + 1ST level. For example, given



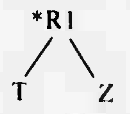


and the triplet set is,

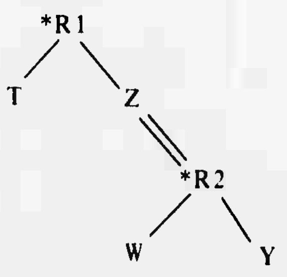
(T,Z,*R1)
(W,Y,*R2)
(W,Z,*R5)
(X,W,*R5)
(Y,Z,*R5)

The triplet set has been shown to reduce to (T_0,Z_1,*R1), (W_2,Y_1,*R2), (X_2,W_2,R5). One parsing of the figure is developed as follows:

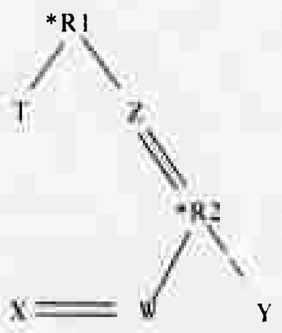
(1)



(2)



(3)



As indicated earlier, the forming of sublevels in the tree structure is the reason for the ordering of the triplet set into subsets. The partial ordering of the triplets within the subsets is to facilitate the suppression of triplets while a tree structure is being formed. The suppression of the triplets is necessary because the triplets indicated by either of the triplet subsets below (where $i,j \neq 5$), contain redundant information.

(i) $(\alpha,\beta,*Ri)$, $(\alpha,\gamma,*Ri)$, $(\beta,\gamma,*Rj)$
(ii) $(\alpha,\beta,*Ri)$, $(\gamma,\beta,*Ri)$, $(\alpha,\gamma,*Rj)$

Any two of the three triplets of (i) or (ii) completely determine the third triplet. The redundancy is evident since the third triplet is often added to the triplet set because a missing relation is determined by the completeness tests. The added triplet provides additional parsings of the figure.

The partial ordering of the triplets of the subsets which comprise the triplet set place subscripts on the triplets which satisfy either of the above subsets. To correctly use the partial ordering within the subset, any two of the three triplets may be used in forming the tree structure and the third triplet is suppressed. Since the order in which the two chosen triplets are used determines different parsings, there are six possible parsings using one of the above triplet subsets. Whether all six parsings are distinct depends on the particular figure being parsed. The six possible parsings of the three triplets of triplet subset (i) are listed in Figure 8. A set of six similar parsings can be obtained from the triplet subset (ii). The subscript on the i and j are to distinguish from which triplet each syntactic relation came.

While the arguments for the syntactic relations which were described in Chapter 3 are not being shown in the examples, it is important to note that their values do change as they enter the tree structure. The difference between two parsings may only be the difference in value of the arguments of a single syntactic relation appearing in the tree structure. Parsings (1) and (3) of the above example illustrate this fact. The difference between parsings (2) and (4) are also only the difference in the value of the arguments of $*Ri_{\alpha\beta}$ in (2) and $*Ri_{\alpha\gamma}$ in (4).

The syntactic relations found in the triplets transfer directly to the tree structure but the value of their arguments do not. The arguments change in value because, while the higher level constituents are being related by the same syntactic relations which originally related the primitives of the figure, the higher level constituents have different dimensional values than the primitives and hence require different values for the arguments of the same syntactic relations.

The obtained syntax tree can be considered a rooted tree. A rooted tree is a tree in which one node, called the root, is given a special significance. This introduces a direction in the

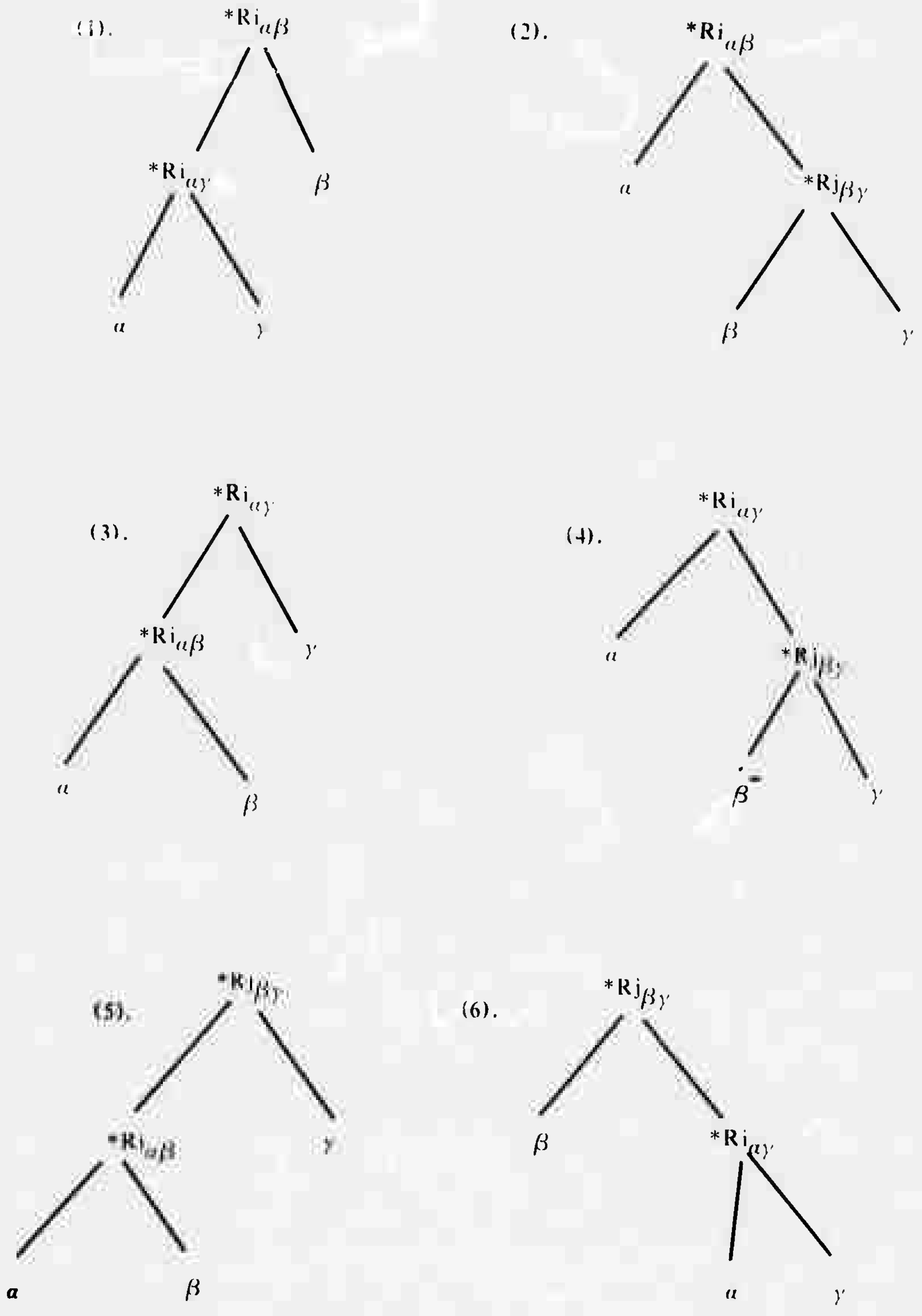
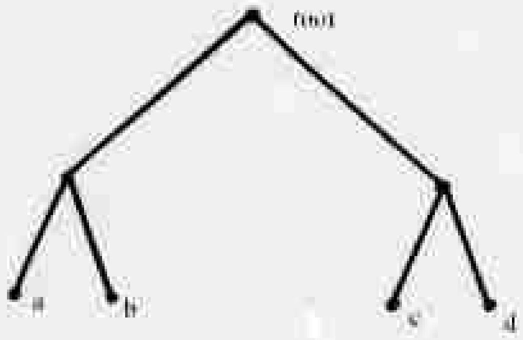


Figure 8. Possible Parsings of a Triplet Subset

tree; away (or in this case down) from the root and towards (up) the root. If a tree is rooted each node has a node immediately above it, unless the node is the root of the tree. In addition, because there is a specified order of the lines around any node, the tree is considered ordered



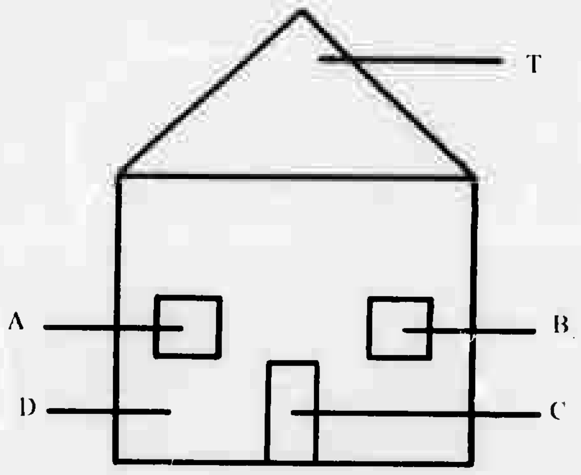
where the nodes are ordered from left to right. A property of the node to be used later, called the *outer degree* of the node, can be determined by drawing paths from the root to the leaves of the tree. This uniquely associates a direction with each arc. The outer degree of a node x is independent of the orientation of the tree.

5.3.4 THE REVERSE POLISH STRING

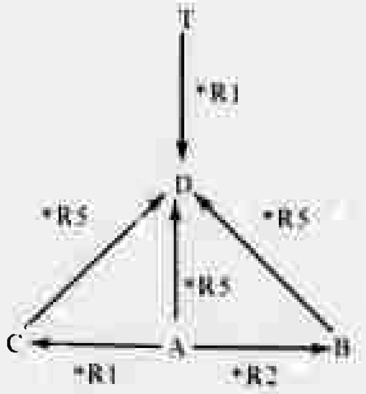
The tree structure is not the desired final form of the structural description. It will be converted to a linear string which uses Reverse Polish notation to provide a more *useable* form of the structural description to the semantic component. As pointed out earlier, the explicit formation of the tree structure is not a necessary step in order to obtain the Reverse Polish string.

The tree structure can be readily transformed into a linear string which uses Reverse Polish notation to order the constituents and the syntactic relations between them. The string can be written by traversing a path around the tree structure from left the right. The string is written from left to right, where primitives are pulled off as they occur and syntactic relations are placed in the string at their last possible encounter with the path. The one exception is in leaving a sublevel, in which case the *contained within* relation is entered into the Polish String after a primitive of the next higher level is entered into the string. Recall that the *contained within* relation is represented in the tree structure by a double line.

An example should clarify any confusion on this process. Given the figure,



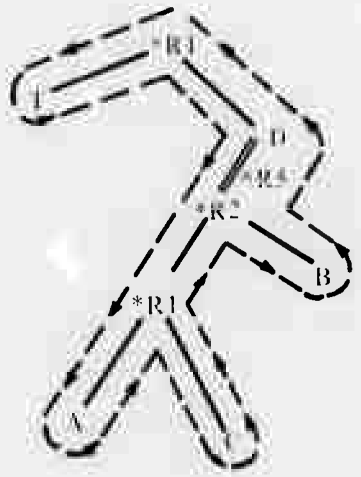
The graph and triplet set is:



(T,D,*R1)
(A,D,*R5)
(B,D,*R5)
(C,D,*R5)
(A,B,*R2)
(A,C,*R1)

Subset 1

One possible parsing of the figure is,



The Reverse Polish string of this parsing is built up from the dotted path giving,

T,A,C,*R1,B,*R2,D,*R5,*R1.

At this time no attempt is made to obtain all possible parsings. Otherwise, a triplet relating B and C would be needed though it would be suppressed in this particular parsing. As mentioned earlier, it is not clear whether every possible parsing is needed for pattern recognition or whether producing every parsing is merely a waste of computer time. The number of different parsings which can be semantically interpreted by the program depends on the degree of extensiveness of the grammar.

In order to obtain the syntactic string directly from the triplet set, the chaining process which operates on linked triplets is used. The chaining process replaces a primitive in the string with the syntactic relation and corresponding primitives of a triplet which is linked to the triplet containing the primitive being replaced. As is shown in forming the tree structure, partial ordering and suppression of triplets in the triplet set are necessary.

Using the example on this page, the steps in forming the syntactic string are as follows:

(1) T,D,*R1
(2) T,A,D,*R5,*R1
(3) T,A,B,*R2,D,*R5,*R1
(4) T,A,C,*R1,B,*R2,D,*R5,*R1

In order to test the well-formedness of a syntactic string in Reverse Polish form the procedure outlined below is followed.

(i) Assign a weight W to each element of the string. The weights are assigned by the formula $W = 1 - d$, where d is the number of lines leaving a node (away from the root as defined in 5.3.3).

(ii) Find $\sum W(i)$ for $i = 1,\ldots,n$ elements of the string, where the left most element of the string is $i = 1$.

(iii) Well-formedness requires that $\sum^{n} W(i) = 1$ and $\sum^{j} W(i)$ for any $j < n$ is never less than 1, i.e., $\sum_{i=1}^{j} W(i) \geq 1$, $j = 1,\ldots,n$.

Testing the well-formedness of the syntactic string of the previous example:

	T	A	C	*R1	B	*R2	D	*R5	*R1
i :	1	2	3	4	5	6	7	8	9
W(i) :	1	1	1	−1	1	−1	−1	1	−1
ΣW(i) :	1	2	3	2	3	2	1	1	1

Since $\sum_{i}^{j} W(i) \geq 1$ for all $j = 1,\ldots,9$ and $\sum_{i}^{9} W(i) = 1$, the string is well-formed. Note that the degree d of *R5 in the tree structure is 0 and the degree of the primitive D is 2.

5.4 A SUMMARY OF THE SYNTACTIC COMPONENT

The syntactic string will now be sent to the semantic component for a semantic interpretation. Because the semantic component is composed of a grammar which operates on linear strings of constituents, the picture which is a two-dimensional configuration must be reduced to a one-dimensional configuration. This reduction is performed by the syntactic component which, in summary obtains the structural description in the form of a linear string by using the grammar K1* to form the lines in the figure from the discrete points and then form the higher level primitives from the lines. Because the syntactic relations are contained implicitly in the pictures but must be made explicit in the syntactic strings for the semantic component, the syntactic component develops the triplet set of 3-tuples which are primitives and their corresponding binary relations. This triplet set is preprocessed and then normalized for the semantic component by writing it as a linear string using Reverse Polish notation.

The transformation of the triplet set actually results in a set of structural descriptions. These multiple parsings can be the result of two different processes.

(1) Because there is a choice as to which artificial lines to add to a figure to allow the figure to be parsed, each different configuration of the figure after the artificial lines have been added provides a different triplet set.

(2) In turn, each triplet set can provide multiple parsings of the figure, as described earlier in this chapter.

Either of the above steps can cause the number of parsings of a figure to be quite large.

The resulting syntactic descriptions are sent to the semantic component, the analyses of which are described in the next chapter.

6. THE SEMANTIC COMPONENT OF SAP

6.1 A SEMANTIC INTERPRETATION

The semantic component operates on the structural description L^*_{ij} to provide a semantic interpretation to the pictorial data, using a grammar to provide this interpretation. As indicated in the previous chapters, a one-dimensional grammar operates on linear strings. For this reason the syntactic component transforms the input picture L_i to a syntactic string L^*_{ij} which structurally describes L_i. The grammar of the semantic component operates on the syntactic string, eventually placing it into one or more classes. The classification of the syntactic string is actually the assigning to it the names (labels) of the classes into which it is placed. The forming of higher level constituents from the symbols of the syntactic string is defined by the rule stated in Chapter 4 for forming segments of syntactic strings. This rule is,

```
<sss>::=<primitive><primitive><syntactic relation>|
        <sss><primitive><syntactic relation>|
        <primitive><sss><syntactic relation>|
        <sss><sss><syntactic relation>
```

This is partially represented by a tree structure in Figure 9. The semantic component, in using the grammar, will assign names of classes to the constituents which are generically represented by <sss> in the tree of Figure 9.

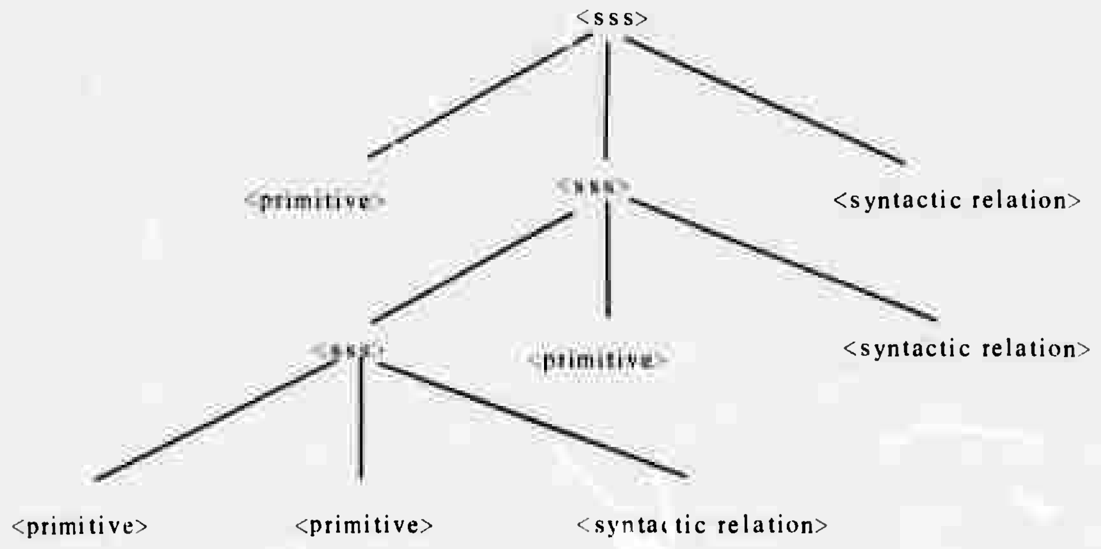
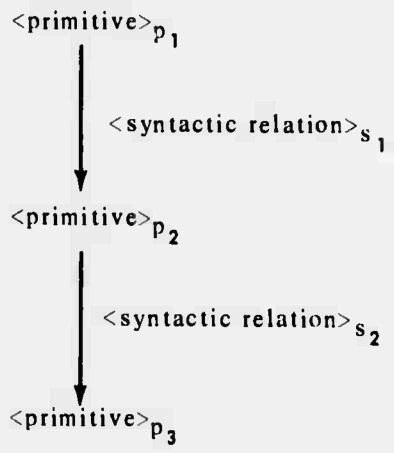


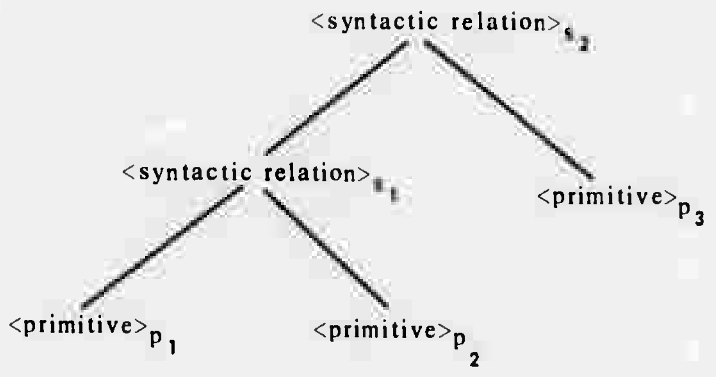
Figure 9. General Syntax Tree

Before describing the grammar which comprises the semantic component, it is worthwhile to first further discuss the syntactic description which the semantic component receives. It was shown in the last chapter that the Reverse Polish string, which is the syntactic description, is obtained from a tree structure which is also a syntactic description of the figure to be semantically interpreted.

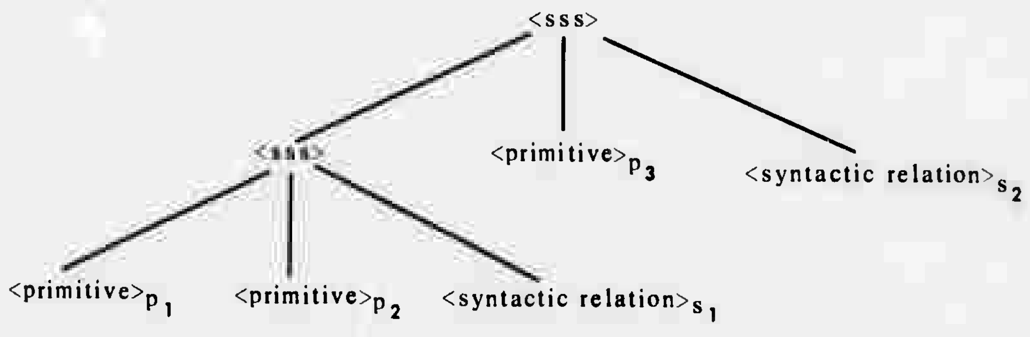
Using the example of the directed graph,



the tree structure obtained by the syntactic component representing one parsing of the figure is of the form shown below.



The important point is that the tree structure immediately above is isomorphic to the tree structure shown below,



the general form of which is presented in Figure 9. However, the <sss> constituents in the above tree structure remain unlabeled by the syntactic component. It is the semantic component which will assign labels to the <sss> constituents.

The algorithm described in Section5.3.3for creating tree structure of the form shown on this page could be easily applied, with only slight modification, to create the tree structure above from the triplet set. Also, the algorithm described in Section 5.3.4 for forming the Reverse Polish string from a tree structure directly applies to the second tree structure above.

In view of this, it is somewhat arbitrary as to which form of tree structure is chosen to represent the parsing of the figure. The second structure has the advantage of explicitly representing the constituents which are to be labeled by the semantic component. However, in this report the first tree structure on the last page is used because it is a more concise notation by implicitly representing the structural constituents.

The grammar of the semantic component K3* is a 5-tuple (T3,N3,R3,C3,SC). The set T3 is the set of terminal symbols and is a union of the set P of primitives and the set S of syntactic relations. The set N3 is the set of nonterminal symbols of the grammar. The set R3 is the rules of the grammar which determine which generic label should be assigned to a particular constituent. The set C3 is a set of matrices which contain constraints for the rules of R3. The goal of the grammar K3* is a scene, SC, and does not appear on the right hand side of any of the rules of R3. The grammar K3* is partitioned into two levels, each level consisting of rules from R3 which assign meaning to the figures of the scene.

The first level of the grammar consist of rules which assign meaning in terms of the structure of each individual figure. The second level accepts those structural descriptions which have not received a singular semantic interpretation from the first level of the grammar. The figures being sent to the second level of the grammar may be ambiguous at this point, that is, more than one meaning has been assigned to the figure. However, it is just as likely that the figure cannot be completely identified by the first level. The second level attempts to semantically interpret the figure in terms of other figures in the scene. Thus, the second level uses the contextual surroundings of the figure, i.e., the syntax of the scene, to assign meaning, while the first level uses the syntactic structure of the figure, i.e., the contextual surroundings of the primitives to assign meaning to the figure. Contextual constraints are considered at both levels of the semantic component grammar. In addition it is desired that the second level grammar use a form of deductive inference decision making. The subcomponents of the semantic component are shown in Figure 10.

To effectively use the first and second level grammars, the syntactic string is first processed by an abstracting subcomponent. The abstracting process described in the next section obtains an abstraction of the figure to be identified. This abstraction eliminates many fruitless attempts by the first level grammar to assign a semantic interpretation to the figure.

6.2 THE ABSTRACTION OF FIGURES

To effectively use the rules of the first level, an abstraction of the structural description is obtained to provide a preliminary classification of the figure without interference from details of the figure. The preliminary classification eliminates the searching through a great deal of the grammar which would not be applicable. Thus, the first level of the grammar would be partially ordered with respect to general classifications of the graphic data. It is reasonable to consider all data to be amendable to some form of classification based on major characteristics of the expected figures.

The abstraction subcomponent receives from the syntactic component the Reverse Polish string representing the structural description of the scene. Thus, the first task of the abstraction subcomponent is to obtain the individual figures of the scene to enable the first level grammar to operate on these syntactic string segments independently. To obtain the syntactic string segments of the scene a modified *bottom-up* strategy is used on the syntactic string operating on the string from right to left.

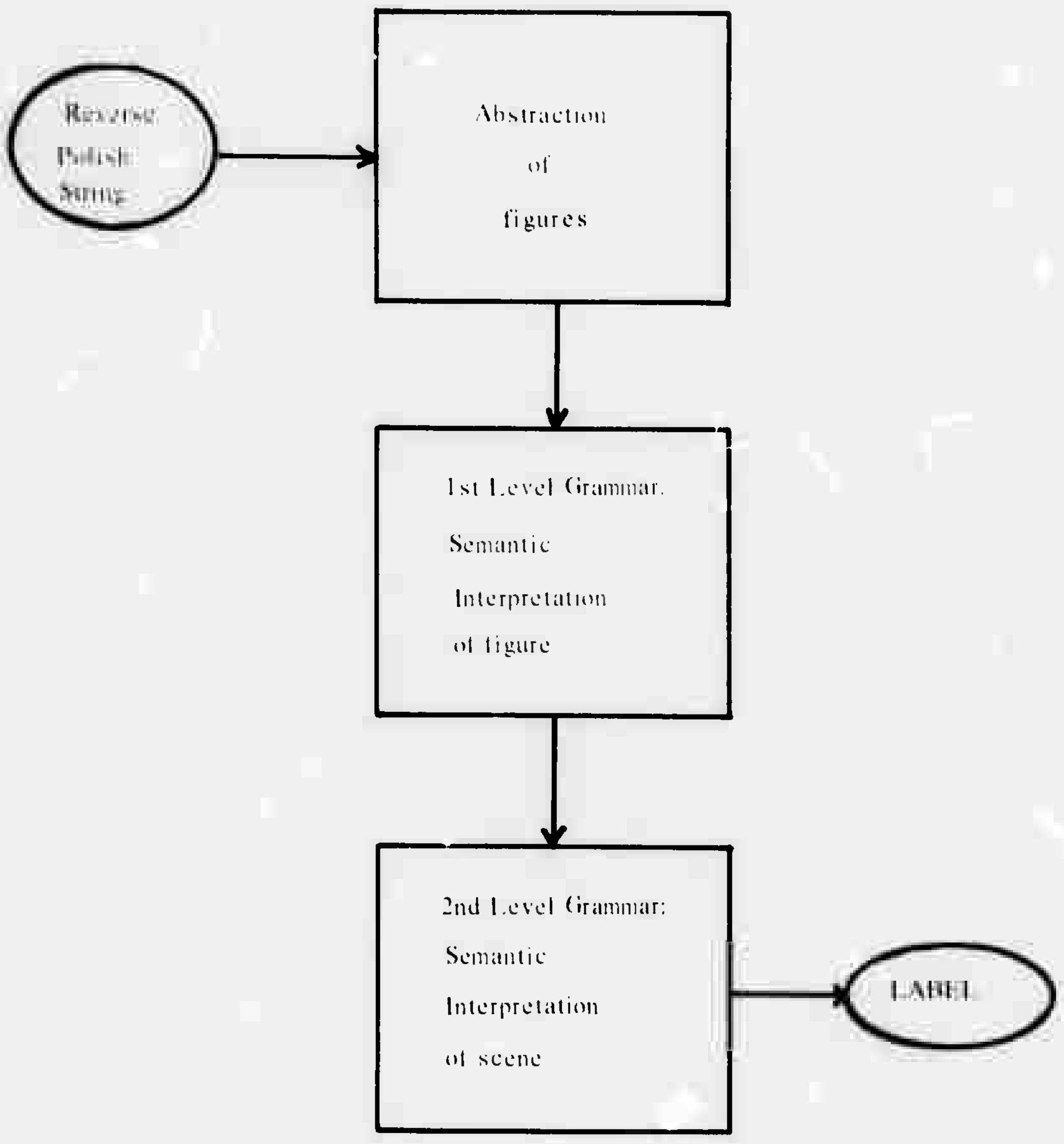


Figure 10. The Semantic Component

The syntactic string is segmented into the various figures according to the grammar rule for forming syntactic string segments. This rule and two others used to obtain the individual figures are listed below.

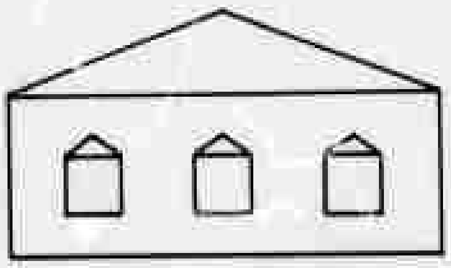
$$
\begin{aligned}
\langle scene\rangle_{ij} &::= \langle figure\rangle_{ij} \mid \langle scene\rangle_{ij}\langle scene\rangle_{ij} \\
&\quad \langle syntactic\ relation\rangle /{*}Ri(n_1>0,n_2,n_3) \\
&\qquad \text{for } i=1,2,3 \text{ or } 4 \\
\langle figure\rangle_{ij} &::= \langle sss\rangle_{ij} \\
\langle sss\rangle_{ij} &::= \langle sss\rangle_{ij}\langle sss\rangle_{ij}\langle syntactic\ relation\rangle \mid \\
&\quad \langle sss\rangle_{ij}\langle primitive\rangle_{ij}\langle syntactic\ relation\rangle \mid \\
&\quad \langle primitive\rangle_{ij}\langle sss\rangle_{ij}\langle syntactic\ relation\rangle \mid \\
&\quad \langle primitive\rangle_{ij}\langle primitive\rangle_{ij}\langle syntactic\ relation\rangle
\end{aligned}
$$

The i and j subscripts on the constituents will be explained shortly. The restriction on the production rule for a scene indicates that a figure is defined to be either of the following:

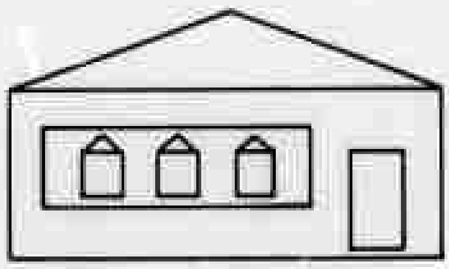
(1) A figure X is represented by a syntactic string segment sss_1 and is related to a figure Y represented by a syntactic string segment sss_2 by a directional relation (horizontal or vertical relation) $*Ri(n_1,n_2,n_3)$ where $i=1,2,3$ or 4 and $n_1>0$. Since $n_1=0$ signifies the two figures X and Y are touching, $n_1>0$ requires X and Y to be two related constructions in a picture which have no contiguous elements. As defined in Chapter 4, n_2 and n_3 indicate the secondary position between constituents, where $n_3=0$ and is not present for a directional relation.

(2) A figure X is represented by a syntactic string segment sss_1 and is related to a figure Y represented by a syntactic string segment sss_2 by a *contained within* relation, $*Ri(n_1,n_2,n_3)$ where $i=5$ or 6.

The two above definitions allow figures to be drawn in a completely recursive manner. Thus, the first definition allows any number of non-contiguous figures in a horizontal row, vertical column, or combination of the two to be structurally represented in a syntactic string and the abstracting component will be able to segment the string into the individual non-contiguous figures. Similarly, the second definition allows any number of figures to be imbedded within a figure. For example, the use of the two definitions allows a row of houses to be imbedded within a house, as shown below.



Or, a row of houses can be imbedded in a subconstituent of the house, such as,



If it is desired to not allow figures to be imbedded within figures, it is necessary only to require the syntactic relation which occurs on the right-hand side of the rule defining <scene> to be a directional relation and eliminate the second definition of above.

Once the syntactic string has been segmented into the segments representing the figure of the scene, the abstracting subcomponent then abstracts the syntactic string segment to obtain the basic features of the figure. The abstracting process in effect first removes all detail from the figure. This leaves an outline of the figure which is further reduced to a set of abstractions of the figure, each abstraction somewhat more abstract than the last. In attempting to generally categorize the figure, the abstractions are used in the reverse order of which they were produced, the most abstract first. The method for performing the abstractions is actually a heuristic which is independent of any grammar being used to semantically interpret the figure.

To obtain the abstractions of the figure from the structural description of the figure, as in obtaining the string representing the individual figure, a bottom-up strategy is used on the syntactic string. The rule to perform this abstracting is the same rule shown above for forming the syntactic string segment constituents. The i and j are attributes of the <sss> constituent which allow the abstractions to take place. As the <sss> constituents are formed, they are assigned the name of the predominant constituent of those constituents which form the <sss>. This predominant constituent is the one which has the largest area. Thus, the i subscript on the <sss> constituents is the name of the predominant subconstituent, where i may be R, TI, etc.. The value of i is then used to name higher level constituents when the particular constituent is considered the predominant constituent. The j is the area of the constituent and is used to determine which constituent is the predominant constituent in forming a higher level constituent.

The following example should clarify the process of abstracting a single figure. The syntactic string segment constituents are represented in the string by Sn(i,k,m) which is defined as: n = segment constituent number: i=1 if the predominant constituent is a rectangle, i=2 if the predominant constituent is an isosceles triangle; k = horizontal dimension; m = vertical dimension. Step 1 is the syntactic string of the figure.

1. R(3,2,2),R(4,2,4),*R4,R(5,2,2),*R4,TI(1,8,6),*R1,R(2,4,5),R(1,8,12),*R5(BOT),*R1
2. R(3,2,2),R(4,2,4),*R4,R(5,2,2),*R4,TI(1,8,6),*R1,S1(1,8,12),*R1

   from S1(1,8,12) $\longrightarrow$ R(2,4,5),R(1,8,12),*R5(BOT)
3. S2(1,4,4),R(5,2,2),*R4,TI(1,8,6),*R1,S1(1,8,12),*R1

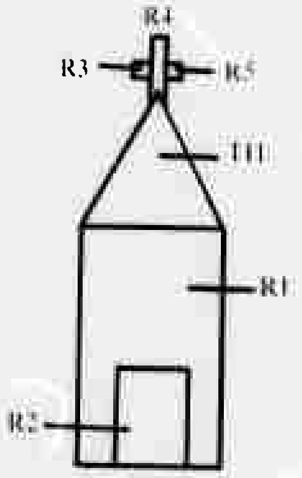
   from S2(1,4,4) $\longrightarrow$ R(3,2,2),R(4,2,4),*R4
4. S3(1,6,4),TI(1,8,6),*R1,S1(1,8,12),*R1

   from S2(1,6,4) $\longrightarrow$ S2(1,4,4),R(5,2,2),*R4
5. S4(2,8,10),S1(1,8,12),*R1

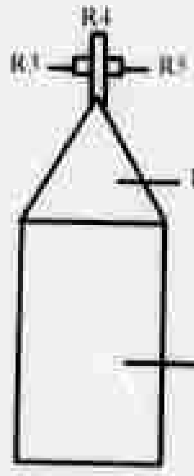
   from S4(2,8,10) $\longrightarrow$ S3(1,6,4),TI(1,8,6),*R1
6. S5(1,8,22)

   from S5(1,8,22) $\longrightarrow$ S4(2(8,10),S1(1,8,12),*R1

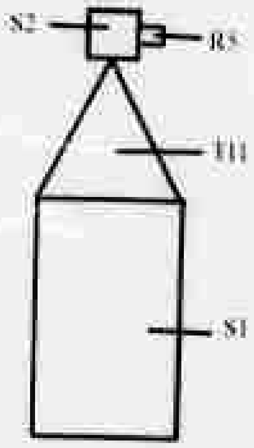
Looking at the abstractions pictorially, 1.



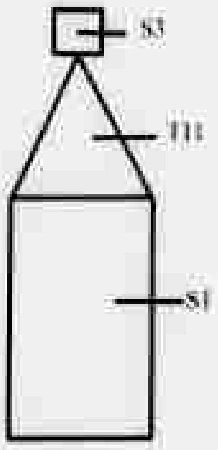
2.



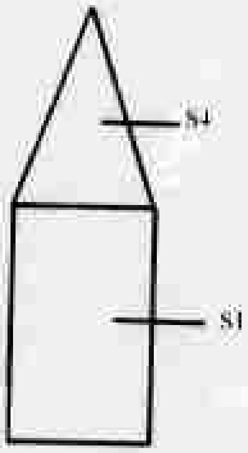
3.



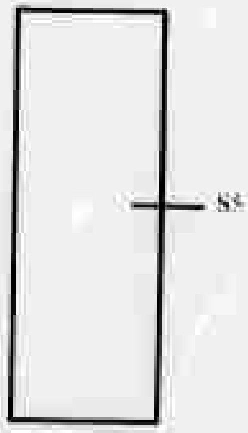
4.



5.



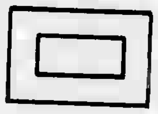
6.



Examining the abstractions in their reverse order, that is the most abstract first, the fifth abstraction or Step 5 would indicate that the figure should be operated upon by the first level grammar and considered to be in a class which represents the category of *house type*.

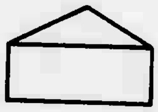
The six categories into which the pictorial data being considered in this report may be placed after it has been abstracted to a string composed of the three ordered constituents, <syntactic string segment>, <syntactic string segment>, <syntactic relation> are listed below.

1. STORE:



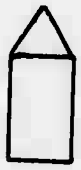
Rectangle contained within rectangle and the final abstraction is a rectangle.

2. HOUSE:



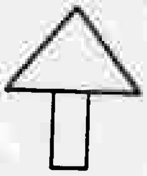
Triangle above rectangle, where 3* (area of triangle) $\geq$ (area of rectangle) and the final abstraction is a rectangle.

3. SILO:



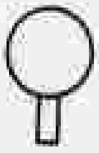
Triangle above rectangle, where 3* (area of triangle) < (area of rectangle) and the final abstraction is a rectangle.

4. TREE:



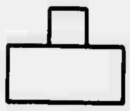
Triangle above rectangle, where (area of triangle) > (area of rectangle) and the final abstraction is a triangle.

5. BULBOUS:



Circle above rectangle, and the final abstraction is a circle or rectangle.

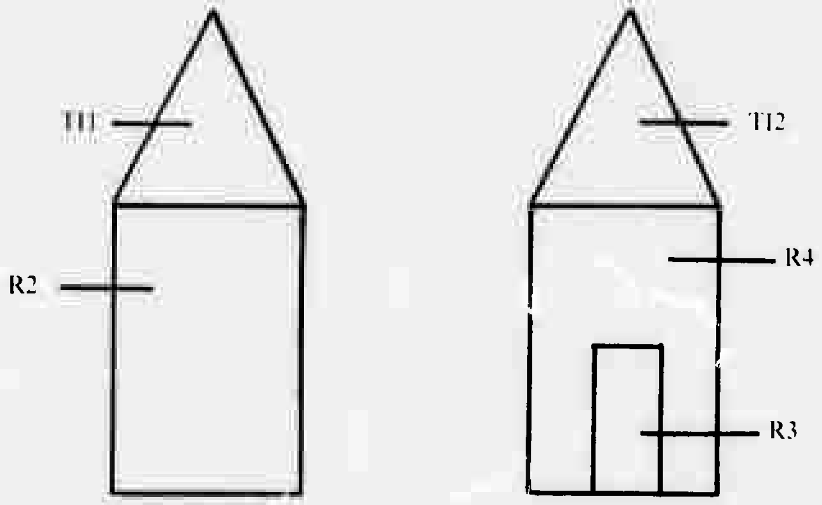
6. OFFICE BLDG:



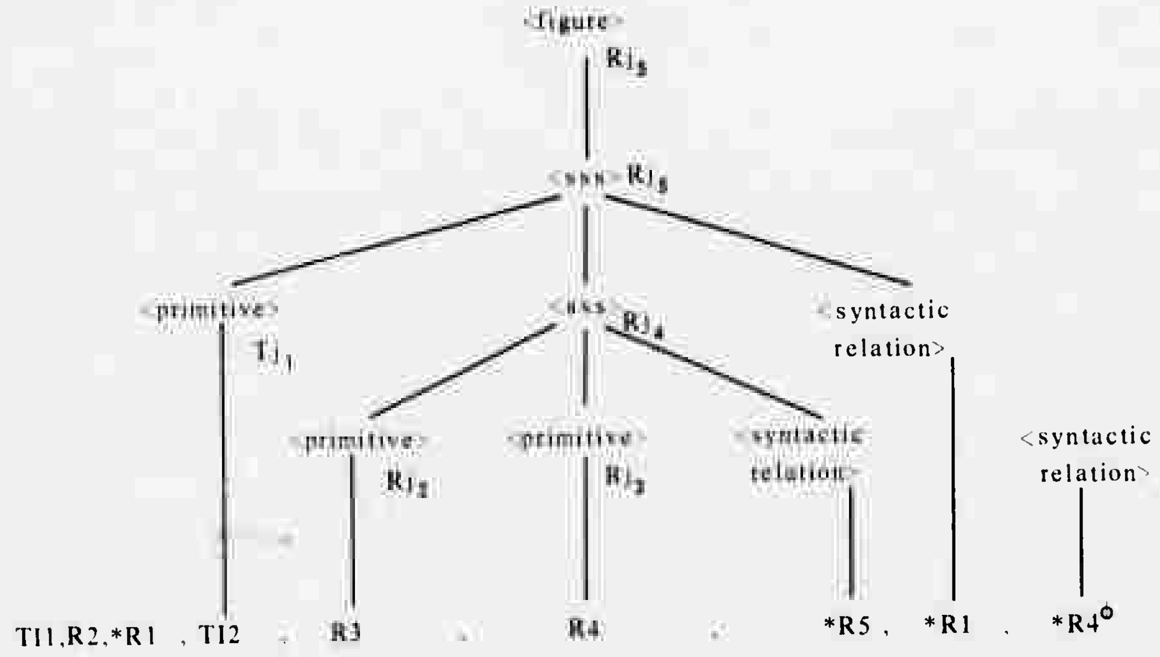
Rectangle above rectangle, and the final sbstraction is a rectangle.

7. UNICORN: Anything which cannot be placed into one of the first six categories.

Because the segmenting of the syntactic string into segments which represent individual figures in the scene and the abstracting process of the figures both use a bottom-up strategy, the two processes can be performed at the same time. Thus, while a scene is divided into the figures which comprise it, the figures are being abstracted. The use of the bottom-up strategy avoids the left recursion problem of the top-down strategy described in Chapter 2. For example, consider a scene composed of the two figures below.



The abstraction subcomponent would operate on the syntactic string to obtain the tree structure shown on the next page. The tree structure indicates the individual figure and its abstractions. As defined earlier, the j subscript on a constituent is its surface area. The relations and primitives do not have their arguments shown. The relation *R4$^{\phi}$ represents the restricted relation $*Ri(n_1>0,n_2,n_3)$ previously discussed in this chapter. Again, the $n_1>1$ indicates that the figures are not contiguous which in turn allows the abstracting component to separate them.



The entire abstracting subcomponent has been implemented in FORTRAN on the IBM 7072. The program is described and listed in Appendix 10.1 of this report.

The abstracting process is a valuable tool in dealing with the problem of multiple parsings. If all of the parsings must be operated upon by the semantic component, the abstracting process should have a great deal of time in eliminating some unproductive attempts by the first level grammar. The multiple parsing problem is also the reason for the semantic component being divided into two levels. The division allows the syntactic string to be semantically interpreted, one segment (which represents one figure) at a time. If each figure was not processed separately, because of the large number of parsings per figure, the computer could possibly run out of space, or use an excessive amount of time. For if a scene of two figures which have m_1 and m_2 parsings respectively, the maximum number of passes through the semantic component 1st level grammar to identify the figures in a scene is $m_1 + m_2$. However, the the number of passes through the semantic component grammar if the entire scene must be considered at each pass is $m_1 \times m_2$. The fact that the abstracting process is a heuristic independent of the grammar rules which it uses may prove to be an additional nicety of the procedure, allowing various criterion other than *largest area* to determine the predominant constituents.

The abstracting process provides a Gestalt of the figure and hence plays an important role in human pattern recognition. M.D. Vernon[42] points out that the most important feature in human pattern recognition is the general outline or contour of the object. This may be because a child learns first by touch and thus associates objects with their contour. In any case there is little doubt that an adult tries to first classify a figure generally before he proceeds to identify it. Few individuals would try to identify a figure as a church if they didn't think that the figure is some type of building. Another manner in which psychologists might describe the abstracting process is providing a *set*. That is, the details of the figure make sense only because we know what type of details to expect by means of a set orientation (no connection to set theory). The abstracting process allows this orientation to take place.

A significant use of the abstracting process is in the filtering of a complex scene. For example, if aerial photographs are to be scanned for a particular figure, the abstracting process would quickly eliminate those figures from the scene which are of no interest. This avoids the brute force approach of attempting to identify every figure in the photograph. Of course, the choice of criterion by which to perform the abstracting is critical.

Once the abstracting process is completed, and by means of one or more of the abstractions the figure has been roughly classified, the original structural description (the complete syntactic string segment) is operated upon by the first level of the grammar.

6.3 THE FIRST LEVEL GRAMMAR OF THE SEMANTIC COMPONENT

As pointed out in Chapter 2, the primitives have no meaning independent of their use in a particular environment. This problem has not arisen in the work by Narasimhan because the primitives are lines as are the graphic data. Hence, except for noise, if a primitive is present it is known to be a bubble chamber track. The severe semantic constrictions on his problem provide the primitives with a singular meaning. But in a less semantically constricted problem where the primitives may have one of several meanings, a major difficulty atises. The problem is that a single primitive, which is able to have multiple meanings, can be assigned a meaning only in the context of other primitives. For example, a rectangle cannot be identified as either a door or a window until its location and dimensions relative to the surrounding primitives are considered.

In view of this difficulty, many of the rules of the first level grammar of the semantic component are context sensitive, providing restrictions which must be satisfied before the rules are considered applicable. These restrictions are generally concerned with the relative size or location of the constituents of the rule. The rules are of the general form,

$$X::=Y,W,{*}Ri,Z,{*}Rj,\ldots/\emptyset$$

$$\emptyset_i = \begin{cases} F(X_k) \quad \Theta \quad G(x_m) \\ x_k, x_m, {*}Rj(n_1,n_2,n_3) \end{cases}$$

where the arguments of the constituents are not shown. Note that the right hand side of the rule is written in early Reverse Polish notation. The rule states that constituent X may be formed from constituent Y in relation Ri with constituent W, this in relation Rj with constituent Z, etc., within the context of $\emptyset$.

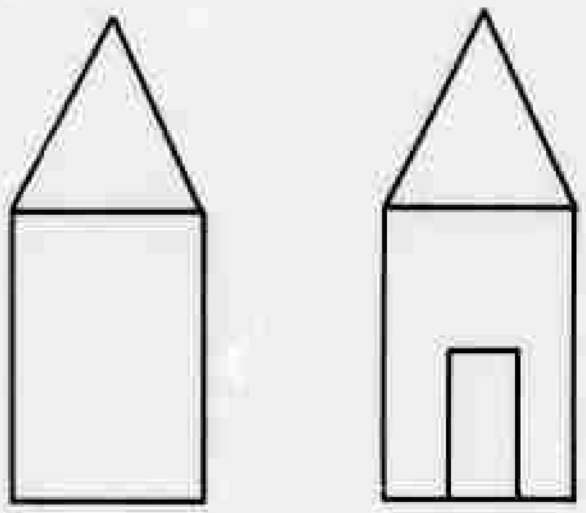
The $\emptyset$ is a set of contextual constrictions $\{\emptyset_i\}$ such that the members of $\emptyset$ may be of either form indicated.

1. When $\emptyset_i = F(x_k)\ \Theta\ G(x_m)$, F and G are functions which obtain the height (H) or width (W) of the constituent x_k and x_m where k may equal m. The Θ is a member of the set of arithmetic relations. For example, if F=H, and x_k=R (a,b,c) then $F(x_k)$=H(R(a,b,c))= c.
2. When $\emptyset_i = x_k, x_m, {*}Rj\ (n_1, n_2, n_3)$ the j, n_1, n_2, and n_3 may be constrained values indicating restrictions on the syntactic relation between the constituents x_k and x_m. The definitions of n_1, n_2, and n_3 are found in Chapter 4.

Thus the constraints may operate on a single constituent or relatively between two constituents. The use of the contextual constraints raises a major problem. The problem is that a single constraint may not necessarily operate on constituents of a single rule but across rules on constituents at different levels. This requires that the formulation of the restrictions be defined and noted independently of the formulation of the rules.

To accomplish this the constraints are written as a set of matrices. C3. Each matrix contains the constraints for different constituents which are restricted by a common constraint. It is then possible for the rules to be flagged with the entries in the matrix which pertain to the constituents of the rules which are to be constrained. While the constraints are between constituents they actually determine the identification or well-formedness of a figure, depending on whether the grammar is being used for recognition or generation. Since it is the figure for which the constraints are ultimately operating and because a top-down strategy is used by the first level of the semantic component, the matrix entries which apply to the constituents of a figure are attached to the rule which has the name of the figure on the left-hand side of the rule.

An example should clarify the preceding paragraph. Consider a grammar which wishes to be able to recognize either of the two structures shown in their general form.



Assume that the structures can be identified as either a doghouse, shed, or garage depending upon how they satisfy various constraints. To perform this recognition the following grammar is used. It should be noted that the use of the grammar is to assign labels to the constituents of the parsing of the figure. Thus, the semantic grammar is essentially assigning labels to the <sss> constituents defined earlier, the assignment algorithm being performed in a top-down manner.

```
<type 1 bldg>::=<doghouse>|<shed>|<garage>
  <doghouse>::=<facade 1> / (1,1),(1,2),(2,2),
                            (2,3),(3,2),(3,3)
      < shed>::=<facade 1> / (1,1),(1,2),(2,2),
                            (2,3),(3,2),(3,3)
    <garage>::=<facade 1> / (1,1),(1,2),(2,2),
                            (2,3),(3,2),(3,3)
   facade 1>::=>roof><front 1><vertical relation 1>
    <front 1>::=<panel>|
                <door><panel><contained within relation 5>
```

```
             <roof>::=<isosceles triangle>
            <panel>::=<rectangle>
             <door>::=<rectangle>
        <rectangle>::= R(n,h,v)
<isosceles triangle>::=TI(n,h,v)
```

The contextual restrictions are indicated by the matrix entries (i,j), which in this case apply to the *type 1 bldg* constraint matrix. The rows and columns of the matrix are labeled with the names of the constituents which are restricted by a constraint θ_i. The names of the rows and columns of the constraint matrix and the corresponding number of the rows and columns are:

roof	= 1
panel	= 2
door	= 3

The actual constraint matrix is shown in Table 2. The entries in the matrix may be defined as follows:

1. For entry (i,j), if $i=j$, then the constraint involves only one constituent and is constraining the constituent's height relative to its width.
2. For entry (i,j), if $i<j$, then the constraint involves two constituents and is constraining some combination of their heights and width.
3. For entry (i,j), if $i>j$, then the constraint involves the syntactic relation between two constituents and is constraining their relative position in one or both directions.

The row and column labels are those constituents which appear on both the left-hand side and the right-hand side of the rules. In addition, constituents which are not explicitly constrained need not be assigned a row and column of the matrix.

To use the constraint matrix, it is necessary to check the list of constraint entries found after the rule, in which the left hand member is the first subgoal being used in the top-down strategy. Thus, one rule generally lists all the constraints for an identification. Each time a rule is used in the identification algorithm the list of constraint entries is checked to see if any are pertinent. If any constraints are to be considered, the correct constraint is obtained from the matrix and performed. If the constituents do not satisfy the constraint, the predicted identification is rejected and a new identification is begun in a top-down manner. If the constraint is satisfied, the present identification continues.

An example using the grammar rules just described and the constraint matrix in Table 2 is indicated in Figure 11. This applies a semantic interpretation to the syntactic string in the form of the label <doghouse>. The complete process would then attempt to apply the remaining labels to the syntactic string. In Appendix .9.2 is contained a larger grammar for pattern recognizing two dimensoinal buildings. The constraint matrices are also included.

All possible semantic interpretations are obtained for the figure. Thus, if in using the top-down technique the figure is identified, the process does not stop until all possible identification are tried. If the figure receives a singular identification, the segment of the syntactic string of the scene which represents this figure is assigned the label or identification which has been obtained. If an additional figure is contained in the scene another segment of the syntactic string representing a second figure is sent to the first level grammar of the semantic component. Again the procedure for semantically interpreting the figure is followed.

Table 2. Example of First Level Grammar Constraint Matrix

Doghouse (DH)
Shed (SH)
Garage (GG)

	roof	panel	door
roof	DH: } SD: } $H(1) \leq W(1)$ GG: $2H(1) < W(1) < 5H(1)$	DH: } SD: } $W(1) = W(2)$ GG: $4H(1) \geq H(2) \geq 2H(1)$	
panel		DH: } SD: } $W(1) \leq H(1) \leq 1.5W(1)$ GG. $H(1) < W(1) < 3H(1)$	DH: $2W(3) > W(2)$ SD. $H(3) \simeq H(2)$ GG: $1.5H(3) > H(2)$
door		DH: } SD: } GG: } *R5(0. BOT. VC)	DH: $H(3) = W(3)$ SD $H(3) \geq 2.5W(3)$ GG. $W(3) > H(3)$

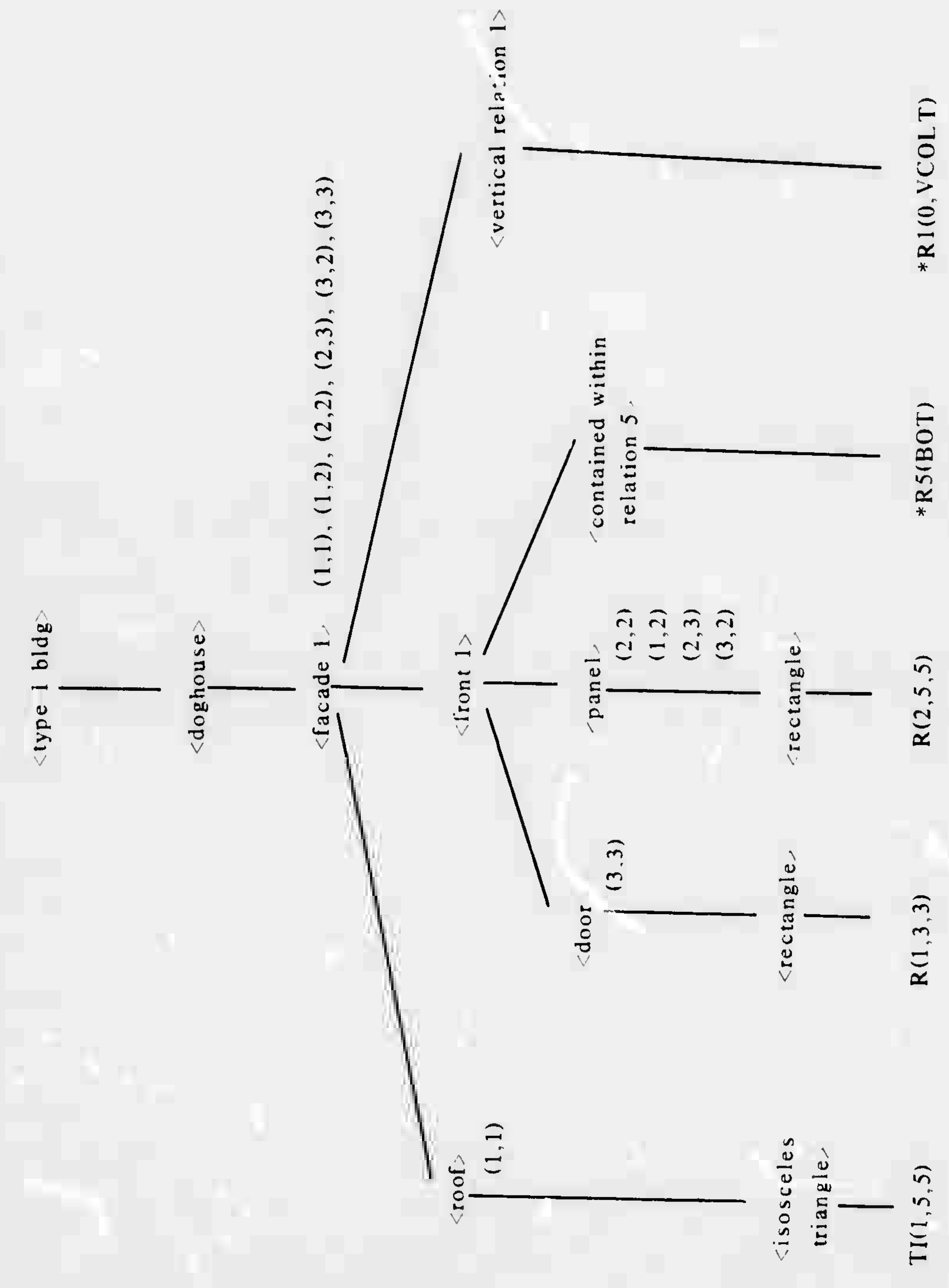


Figure 11. Example of Semantic Interpretation

When all figures of the scene have been processed by the first level grammar, the semantic interpretations of the figures are sent to the second level grammar.

6.4 THE SECOND LEVEL GRAMMAR OF THE SEMANTIC COMPONENT

While the first level works in terms of the syntax of the figure (i.e., the context of the primitives) the second level works in terms of the syntax of the scene (i.e., the context of the figures). Once all the figures of the scene have been operated upon by the first level grammar and the resulting semantic interpretations have been sent to the second level grammar, one of the following situations has been obtained by the first level grammar and the corresponding subsequent steps indicated are taken.

1. Each figure of the scene has received a singular semantic interpretation. In this case the semantic labels are operated upon by the rules of the second level to obtain a semantic label for the scene.
2. Each figure of the scene has received at least one semantic label and at least one figure has received more than one semantic label, that is, the figure is considered ambiguous. In this case the set of all combinations of labels of the figures is processed by the second level grammar. The second level grammar, whose rules define the context of the figures may eliminate some of the semantic interpretations which cause one or more of the figures to be ambiguous. It is possible that after the processing by the second level grammar the figure will still be ambiguous. In this case the scene will receive more than one semantic interpretation, making it ambiguous.
3. At least one figure of the scene has received at least one semantic interpretation and at least one figure has received no semantic interpretation. that is, the figure is considered anomalous. In this case a deductive inference procedure will be used to attempt to semantically interpret the anomalous figures.

The first and second case of above use the same procedure of a bottom-up strategy in applying the rules of the second level grammar. The third case in addition uses a form of deductive inference to semantically interpret the scene. Its procedure is outlined in Chapter 7 under the section on future extensions.

The rules of the second level grammar are of the same form as the first level grammar, that is,

$$X{=}Y,W,{*}Ri,Z,{*}Rj,\ldots/\emptyset$$

where Y, W, and Z are constituents related by the indicated relations. The ∅ is again a contextual constraint and has the same definition as that for the first level grammar. The constraints are placed in matrices as they were for the first level grammar. An example of such a matrix is shown in Table 3, where house=1, garage=2, and doghouse=3. A sample of the second level grammar is as follows:

```
<scene>::=<backyard> | <home>
<home>::=<house> <backyard> <horizontal relation> |
               <house> <garage> <horizontal relation> |
               <house> <doghouse> <horizontal relation> / (1,1),(1,2),
                                                          (1,3),(2,2),(3,3)
```

Table 3. Example of Second Level Grammar Constraint Matrix

Home (HM)
Backyard (BY)

	house	garage	doghouse
house	HM $3W(1) > H(1)$	HM $H(2) = H(1)$	HM $H(1) > 3H(3)$
garage	HM: $1.2.*Ri(n_1.BOT)$	$\left.\begin{array}{l}\text{HM}\\ \text{BY:}\end{array}\right\} 2H(2) > W(2)$	BY $\left\{\begin{array}{l}W(2) > 4W(3)\\ H(2) > 2H(3)\end{array}\right.$
doghouse	HM: $1,2*Ri(n_1,BOT)$	BY $1,2*R(n_1,BOT)$	$\left.\begin{array}{l}\text{HM}\\ \text{BY}\end{array}\right\} H(3) > W(3)$

<backyard>::=<garage><doghouse><horizontal relation>/(2,2),(2,3),
(3,3)

In the second case, where there is the possibility of the scene receiving more than one label, those combinations of labels of the figures which are not well-formed according to the rules of the second level grammar are rejected. The procedure in using the grammar and the constraint matrices is the same as that described for the first level grammar.

The third case requires some form of deductive inference to supply enough information to use the rules of the second level grammar. The procedure described in Section 7.1 is actually developed for the more complex situation in which more than two figures make up a scene.

If the figure cannot be identified by either the first or second level of the semantic component grammar then the figure may either be truly anomalous or the grammar is not satisfactory for the graphic data being considered. An unsatisfactory grammar may be due to the fact that the grammar is incomplete and hence just not extensive enough or the particular grammar being used may be a poor choice for the particular graphic data. A grammar which is thought not to be sufficiently extensive can most likely be corrected without an unreasonable amount of difficulty. If the grammar is considered a poor choice either a new set of primitives and/or syntactic relations is needed to parse the data or a new set of grammar rules using the same primitives and syntactic relations is required.

7. FINAL CONSIDERATIONS

7.1 FURTHER EXTENSIONS

The following list provides some of the many extensions which can be considered for SAP and for syntax-directed models in general.

1. The ability for the model to learn is a natural development in using a grammar. A higher level of learning would allow the model to have some self-organizing or inductive inference capabilities[43,44] in order to extend its grammar as it sees fit, modifying or redefining rules of the grammar when necessary.
2. The use of deductive inference mentioned in section 6.4 is not as ambitious as the preceding extension and thus deserves some outlining. The use of deductive inference can be applied to the situation in which at least one figure of a scene has received at least one semantic interpretation and at least one figure has received no semantic interpretation. As mentioned in section 6.4, the procedure is actually of significant value only if more than two figures comprise a scene. The inference is made assuming an identification of one of the unlabeled figures. Using this assumption, identifications are obtained for various other figures and these identifications are checked for contradictions. If a contradiction (reductio ad absurdum) does arise, the original assumption is known to be wrong and hence that particular meaning is eliminated as a possible identification of the figure to which it was assigned. Using an inference making procedure necessitates detailed bookkeeping so that a distinction can be maintained as to those figures which have been *rigorously* identified and those figures which have been identified on the basis of the assumed identification of another figure. If an assumed identification is found to be incorrect due to the constraints, any identifications based on the incorrect assumption must be removed by a backtracking procedure.

   The above use of deductive inference can be applied to actually perform either of two types of identification. They are:

   (i) Knowing that a particular identified figure is in the scene, an unidentified figure is completely identified.

   (ii) Knowing two partially identified figures in a scene, each is identified from information of the other.

   In the second type of deductive identification situation, the partial identification would come from the first level grammar of the semantic component. The difficulty is in extrapolating these partial identifications to deductive inferences by the second level grammar. This is an interesting problem which has yet to be considered in the literature.
3. The possibility of merging the generation and pattern recognition modes into one totally integrated system provides for further learning capabilities plus a high degree of man/machine interaction.
4. The use of a two-dimensional grammar in man/machine interaction would

allow the user to define grammar rules without needing an extensive knowledge of the system. A recent Ph.D. Thesis by T.A. Standish[45] has indicated an approach to this problem.

5. The use of syntactic analysis to allow dynamic pictures to be generated has unlimited applications.
6. Without a doubt, the most sophisticated extension is the ability to operate with figures which contain information concerning depth. The quantum jump from two to three dimensions would produce problems manyfold, but if pattern recognition is to be of any truly significant value, these problems must be tackled and solved.

7.2 ADVANTAGES OF THE SYNTAX-DIRECTED MODEL

The advantages of the general syntax-directed model for pattern analysis can be described briefly by the following points:

1. What appears to be the strongest point of syntax-directed analysis is its ability to analyze an arbitrarily complex pattern. The ease with which recursiveness can be placed within the grammar allows an infinite number of variations in the input patterns to be identified.
2. The ability to analyze arbitrarily complex figures indicates a high degree of abstractness may be represented by the grammar, allowing a wide range of classes to be defined by the grammar.
3. The use of the grammar allows detailed distinctions to be made as fine as desired.
4. The use of a grammar which decomposes a pattern into its simpler concatenated parts provides not only the name of the pattern but a structural description which efficiently represents the pattern.
5. The descriptive power of the grammar provides the ability to determine topological equivalence among patterns in addition to providing a basis upon which the semantic analysis can be made.
6. The consideration of graphics displays which have dynamic capabilities is provided an interesting approach by the syntax-directed analysis of figures. Because the topological features of the figure are available, it can be readily translated by translating any single constituent of the figure.
7. The use of the topological features to represent a figure is an extremely efficient manner in which to store pictorial data. Rather than store all of the digitized points, the name of a higher level constituent, such as *triangle*, is stored along with its reference point. This reference point and the name of the constituent completely determine the set of points which comprise the constituent.
8. Also because the topological features are available and hence, the constituents of the figure are related, additions to the figure may be made by essentially augmenting the structural description of the figure without altering the already existing structural description.
9. The use of a grammar allows the syntax-directed analysis to be invariant under linear displacement and size.

10. The grammar is easily extendable and may be altered with no ramifications in most cases.
11. The efficacy by which the grammar may be extended provides an opportunity to add sophisticated learning techniques to the syntax-directed models.
12. A further advantage of the ease by which the grammar may be changed allows for a wider range of patterns to be recognized without large changes in the syntactic and semantic components.
13. A strong factor which has just begun to be utilized is the ability of the grammar to operate as either a generator or recognizer of patterns. This allows for a high degree of man machine interaction without a great deal of overlapping operators.
14. The ability of the syntax-directed pattern recognizer to operate in a parallel mode may yet prove to be its strongest point. Using a top-down strategy various alternatives may be considered simultaneously by using a grammar. This should prove to be an enormous time saving factor and also serves as a check for errors in the identification process.
15. Because the syntax-directed analysis processes all the information in an organized and meaningful way, both local and distant Gestalt qualities are obtained. This is particularly appealing to psychologists and physiologists who are r versing the normal research behavior by studying the techniques used in computer programs to hypothesize about the workings of the human conceptual and physical processes.

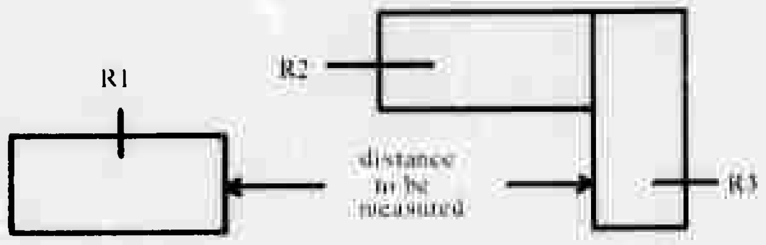
7.3 DISADVANTAGES OF THE SYNTAX-DIRECTED MODEL

The disadvantages of such a model can be described by the following points:

1. Not only the strength, but also the weakness of the model lies in the grammar. The choice of primitives, syntactic relations, and higher level constituents to be formed essentially distinguishes one grammar from another. This choice is critical and for syntactic analysis to reach any degree of sophistication, a formalization for this choice must be developed.
2. What may prove to be a major weakness of syntactic analysis is the multiple parsings which are obtainable for a figure. These multiple parsings, however, do not appear to carry the same degree of ambiguity that they do in natural language analysis though there is a seemingly astronomical number of different parsings that can be obtained from a figure of only reasonable complexity. If, as it appears likely, all of these parsings would receive the same semantic interpretation, then rather than process all of them, the grammar needs to be developed to be able to semantically interpret any one of these parsings. As this solution is infeasible, the syntactic analysis needs to obtain the parsing in a normalized manner and the grammar which provides the semantic interpretation is then designed to expect the parsing in this normalized form. If the multiple parsings of a figure do receive different semantic interpretations then it is necessary to obtain all of them. In this case,

to eliminate some of the parsings it is necessary to define higher level primitives which absorb structural attributes not significant in the pattern recognition process. (This is done in SAP by defining triangles, etc. to be higher level primitives upon which the semantic component operates.)

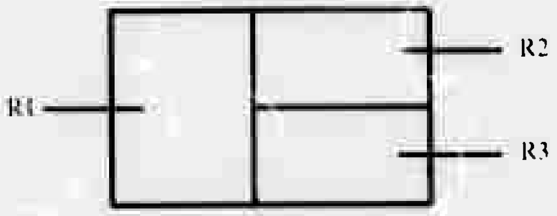
3. The problem of multiple parsings can be considered to create a semantic analysis problem. A second semantic analysis problem is the use of the structural descriptions by the semantic component grammar. The process is considered slow and not efficient because of the number of false starts by either the top-down or bottom-up techniques. Parallel processing should be a solution to this. An alternative to the use of a grammar for the semantic analysis is the use of discrimination nets (decision trees). EPAM[46,47] and other models of this type[48,49] have shown the discrimination net to be an effective decision maker and adaptable to various learning and self-organizing algorithms. While a net does not represent recursion as well as a grammar, it has not been found that the semantic component needs a highly recursive mechanism.
4. It is possible that a grammar will not be able to resolve ambiguities as some ad hoc pattern recognition techniques. One difficulty which has been resolved but continues to cause some anxiety is the measuring of distances between edges. The following two examples should indicate the problem. (i) In the drawing below, the distance of a horizontal relation between primitive R1 and the constituent formed from primitives R2 and R3 is shown. A user might be tempted to consider the distance to be that between R1 and the left edge of R2.



(ii) A possible difficulty in syntactic representation may also be seen by the example

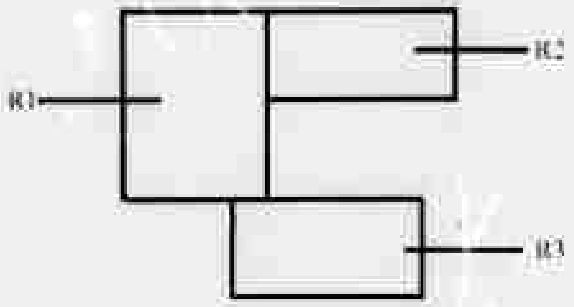
R(1,4,8), R(2,8,4), *R4(TOP), R(3,8,4), *R1(LE4LT)

gives



while

R(1,4,8), R(2,8,4), *R4(TOP), R(3,8,4), *R1(LE3LT)



5. Because most complex graphical data has contextual constraints, the responsibility on formulating these constraints to avoid mislabeling ambiguous or anomalous figures lies on the person creating the grammar.
6. The choice as to the primitives of the grammar is made difficult by fact that the primitives are essentially contained implicitly in the graphic data.
7. The division between hardware and software is a point of detention in syntactic analysis. Syntactic analysis is essentially a software tool while several aspects of the analysis may be performed better by a hardware component. An example is the syntactic description of a circle, which should perhaps be incorporated in the hardware.

7.4 COMPARISON SUMMARY

The above list of advantages and disadvantages of the syntax-directed model for pattern recognition should not be considered independent of the alternative methods for pattern recognition. Using Minsky's[50] classification of pattern recognition models as templet matching, property list matching, and articulate descriptions (syntactic analysis), the advantages and disadvantages may be summarized as follows:

I. Numbers 1,2,4,5,6,8,11,13,15 of the list of advantages of the syntax-directed model are advantages over the templet and property list matching techniques.

II. Numbers 3,7,9,10,12 of the list of advantages are advantages over only the templet matching technique.

III. Numbers 1,2,5,6 of the list of disadvantages are true disadvantages compared to the templet and property matching techniques.

8. SUMMARY AND CONCLUSION

The model SAP presented in this report is a study in the methodology of the syntax-directed pattern recognition and generation of pictures. SAP performs a many-to-one mapping by accepting pictorial data as input and providing as output a label for the data. The pictorial data considered are two-dimensional pictures which contain no depth information.

In order to operate as either a recognizer or generator of pictures, SAP is composed of two components, a syntactic component and a semantic component. In terms of pattern recognition, the syntactic component accepts a picture L_i and translates it to a set of strings of symbols L^*_i which describes the picture's structure. Thus, the function of the syntactic component is translating from a two-dimensional language L to a one-dimensional language L*. The syntactic component uses a lexicon, a modified phrase structure grammar, and a set of transformational rules to perform this translation.

The semantic component accepts structural description L^*_{ij} from the set of structural descriptions L^*_i of picture L_i and attempts to apply a semantic interpretation, designated as Label_{ij}, to the picture. First a set of abstractions of L_i are obtained by various operations on the string of symbols L^*_{ij}, to obtain a general classification of L_i. The string L^*_{ij} is then operated upon by a context sensitive modified phrase structure grammar to receive a semantic interpretation. The recognition process is made in terms of the syntax and context of the figures which comprise the picture.

SAP also has the facility to allow a high degree of man machine interaction. A formalization of the syntax L* allows a user to by-pass the syntactic component and have the semantic component attempt to semantically interpret a structural description which he has created or by-pass the semantic component to generate a picture from a structural description which he has created.

Aspects of the syntactic component of SAP which appear particularly promising are the use of transformation rules to add artificial lines to a picture and the combination of a lexicon and a set of 3-tuples to explicitly represent the implicit syntactic relations which form the topological features of a figure. The language L*, though highly restricted, is capable of representing the structural description of rather complex two-dimensional figures, which in turn allows the user to readily interact with SAP.

Similarly, significant contributions of the semantic component of SAP are the use of an abstracting process to obtain a general classification of the figure and the ability to represent contextual constraints which exist between various levels of the constituents which comprise a figure or scene (the entire picture). These tools are deviations from the use of a phrase structure grammar to obtain the stuctural description and semantic interpretations of a figure or scene. This deviation is the type of direction that must be taken in order to be able to syntactically analyze a complex two-dimensional picture.

In conclusion, the ability of the syntax-directed model to be able to recognize or generate an arbitrarily complex figure, determine the topological equivalence among patterns, readily apply to dynamic displays, provide an efficient approach to storing a pictorial data base, and operate in a parallel mode indicates that such a model is a highly desirable method of operation for dealing with at least some types of pictorial data. Because the syntax-directed model does require a degree of formalization of the structure of the expected pictorial data and there is difficulty in dealing with multiple structural descriptions of a single figure, such a model may not be necessarily applicable to all cases of pattern recognition and generation. However, the ability to extend the syntax-directed model to perform inductive and deductive reasoning and ultimately to process three-dimensional information indicates the high degree of sophistication which the syntax-directed model should be expected to achieve.

APPENDIX 9.1

THE IMPLEMENTATION OF SAP

Various aspects of SAP have been implemented on the LINC, the IBM 360/50, and the IBM 7072. The first implementation, Program I, was on the LINC, a small general purpose digital computer, to study the type of pictorial data chosen. The programming language used is an assembly language, LAP6.

Basically, Program I is composed of subroutines which are able to display any of the geometric primitives for any specified dimensions. The location of the primitives on the screen is determined by giving a particular value to their reference point. The information is entered through the keyboard, and thus by entering a group of referenced primitives, any desired scene may be displayed on the storage scope. The actual notation of the referenced primitives is displayed on the LINC screen, allowing changes and deletions to be made as it is typed into the LINC. The photographs in Figures 9.1.1(a), 9.1.2(a), and 9.1.3(a) are examples of the graphic feedback. The horizon is obtained by referencing a rectangle of zero height. Also, the circles and ellipses are generated using sine and cosine tables, rather than approximated by eight directional lines, as described in Chapter 5.

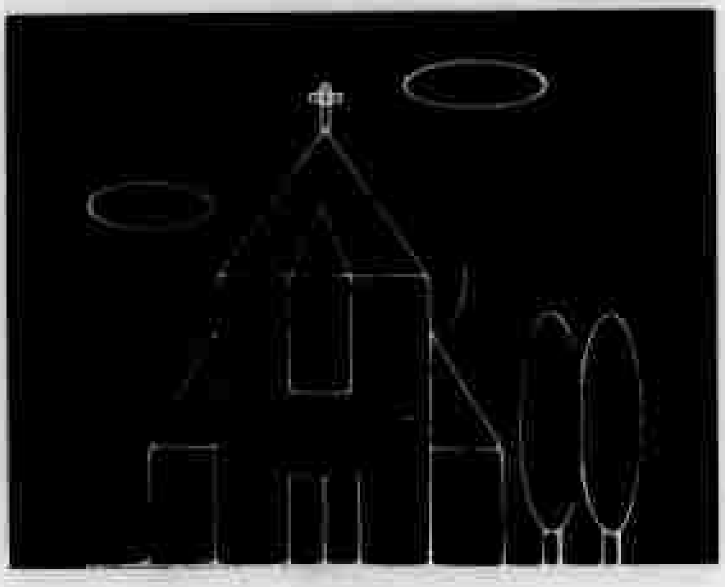
Program II, also on the LINC in LAP6, is a first implementation of L**. Thus, after a syntactic string, L^*_{ij} is typed into the keyboard, the graphic representation of the string is displayed on the storage scope. Using a single referenced primitive in the string, the reference points of the other primitives are obtained by the syntactic relations relating the primitives. This allows the user to see a display of the parsing he has just typed into the LINC. With only slight extension, this same procedure will provide information in the form of graphic displays of the various abstractions performed on the string by the semantic component.

The various options of the graphic syntactic language indicated in Chapter 4 are not allowed at the present time. While more efficient compiling routines could be written to code the input string, this topic is incidental to the development of SAP and has been ignored. In Figures 9.1.1(b), 9.1.2(b), and 9.1.3(b) are examples of the input strings required to produce the scenes shown in Figures 9.1.1(a),9.1.2(a), and 9.1.3(a). It should be noted that the present implementation cannot yet fully generate these graphic scenes from their respective syntactic string. The graphic scenes were produced by the referencing technique of Program I.

A program on the IBM 360/50 has been written in LISP 1.5 to perform various manipulations on the triplet set and transform the triplet set into a Reverse Polish string. The algorithm which obtains the Reverse Polish string from the triplet set bypasses an explicit formation of the syntax tree.

In Figure 9.1.4 is a flow diagram of the program which obtains the Reverse Polish string from the triplet set. Section 9.1.1 contains a listing of the program.

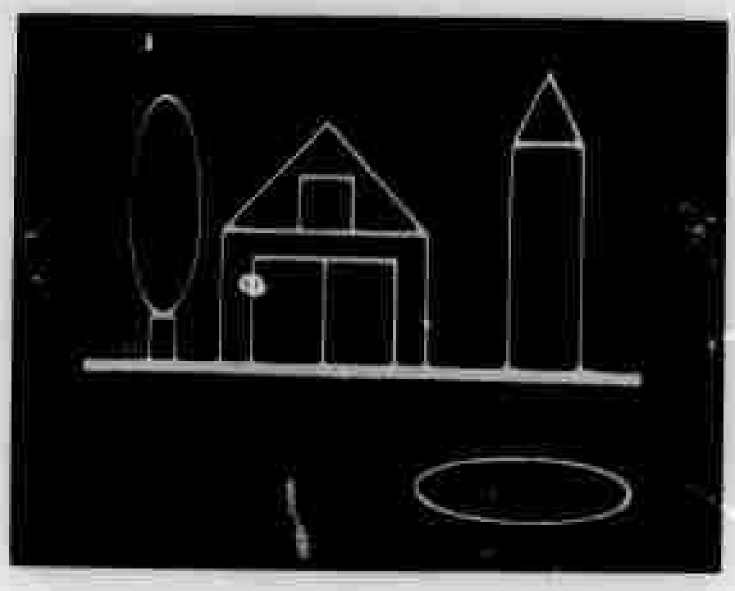
The entire abstraction subcomponent has been programmed in FORTRAN on the IBM 7072. As indicated in Chapter 6, the program accepts as input a string which provides a structural description of a scene. The abstraction subcomponent segments the string into individual figures and abstracts each figure, placing the figure into one of the seven general classes listed on page 92. These classes are described in greater detail in Chapter 6 in the discussion of the abstracting component.



The Graphic Feedback of a Church Scene
Figure 9.1.1(a)

```
TR(1.10,15), R(1,10,15), *R1(LE), R(2.2,1),
R(3,1.5), *R4, R(4,2,1), *R4, TI(1,30,20),
TI(2,7,10), *R5(BOT),*R1, R(5,4,12), R(6,4,12),
*R4, R(7,7,15), *R1(13), R(8,30,40), *R5(BOT),
*R1, *R4(BOT), TL(1,10,15), R(9,10,15), *R1(LE),
*R4(BOT), E(1,17,5), *R3(TOP12UP), E(2,20,5),
*R4(10,TOP1DN), E(3,7,30), R(10,2,4), *R1,
*R4(2,BOT), E(4,7,30), R(11,2,4), *R1, *R4(BOT),
R(0,777,0,0,-350), *R1(RE8LT)
```

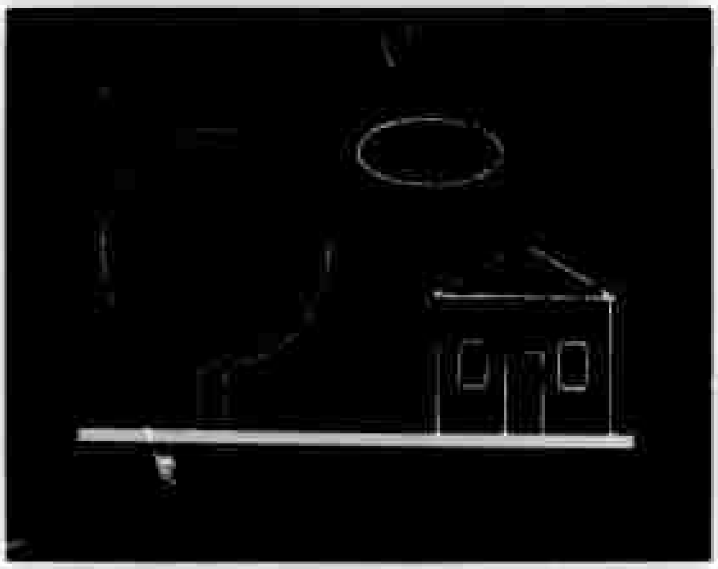
The Syntactic String to Generate the Church Scene
Figure 9.1.1(b)



The Graphic Feedback of a Farm Scene
Figure 9.1.2(a)

```
E(1,10,30), R(1,3,6), *R1, R(2,6,6),
T1(1,27,14), *R5(BOT), R(3,20,15), R(4,20,15),
*R4, R(5,27,20), *R5(BOT),*R1, *R3(4,BOT),
T1(2,10,10), R(6,10,32), *R1, *R3(9,BOT),
R(0,777,0,0,-100), *R1(RE8LT), E(2,30,7),
*R1(10,RF)
```

The Syntactic String to Generate the Farm Scene
Figure 9.1.2(b)



The Graphic Feedback of a House Scene
Figure 9.1.3(a)

```
C(1,32,32), R(1,3,10), *R1, T1(1,25,6),
R(2,3,5), R(3,4,12), *R4(1,BOT3UP),
R(4,3,5), *R3(1,BOT3DN), (5,23,20),
*R5(BOT), *R1, *R3(15,BOT),
E(1,20,7), *R1(7,RT10LT),
R(0,777,0,0,-200), *R1(RT2LT)
```

The Syntactic String to Generate the House Scene
Figure 9.1.3(b)

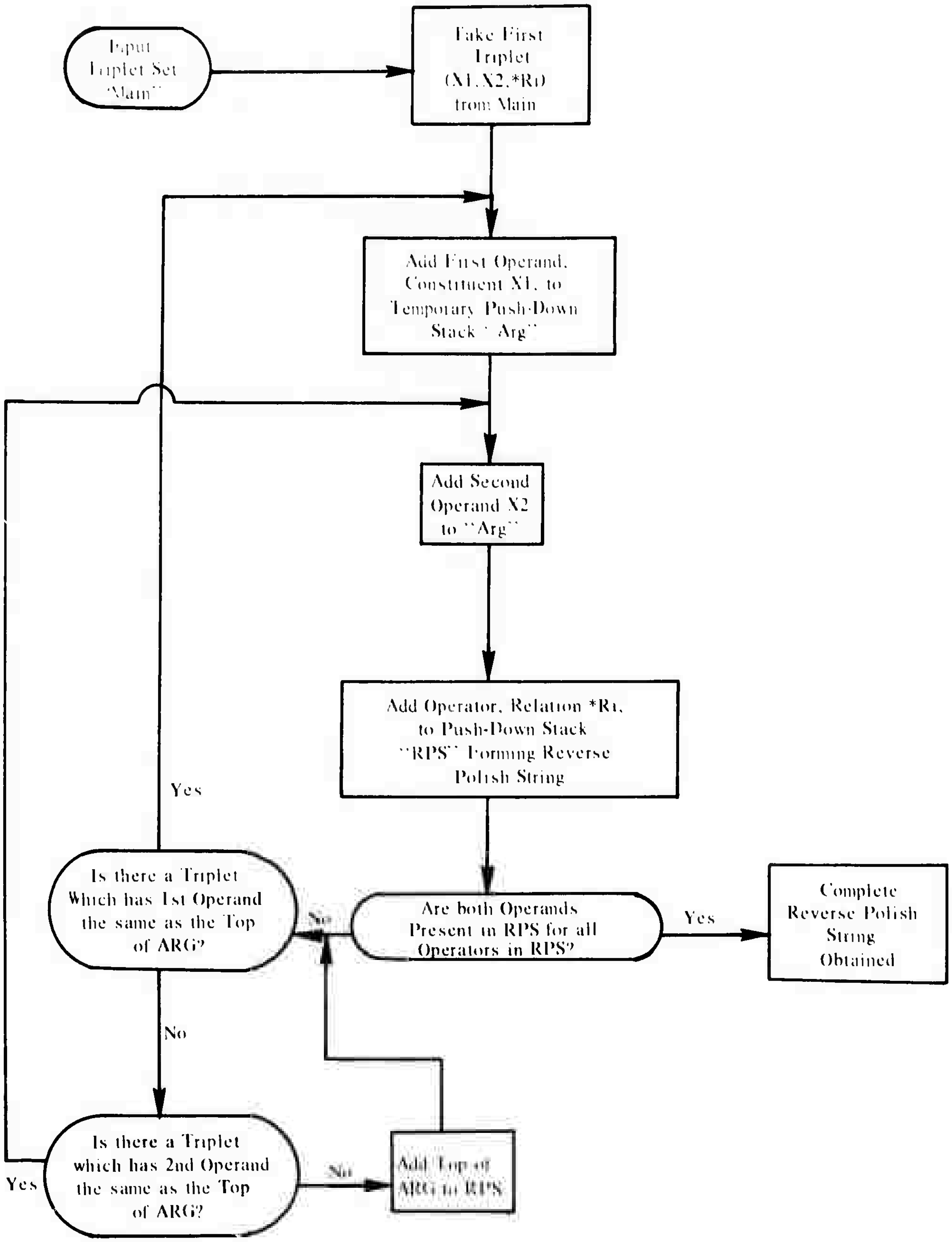

Input Triplet Set "Main"
Take First Triplet (X1,X2,*Ri) from Main
Add First Operand, Constituent X1, to Temporary Push-Down Stack "Arg"
Add Second Operand X2 to "Arg"
Add Operator, Relation *Ri, to Push-Down Stack "RPS" Forming Reverse Polish String
Are both Operands Present in RPS for all Operators in RPS?
Yes
Complete Reverse Polish String Obtained
No
Is there a Triplet Which has 1st Operand the same as the Top of ARG?
Yes
No
Is there a Triplet which has 2nd Operand the same as the Top of ARG?
Yes
No
Add Top of ARG to RPS

Figure 9.1.4 A Flow Diagram of the Program which Obtains a Reverse Polish String From the Triplet Set

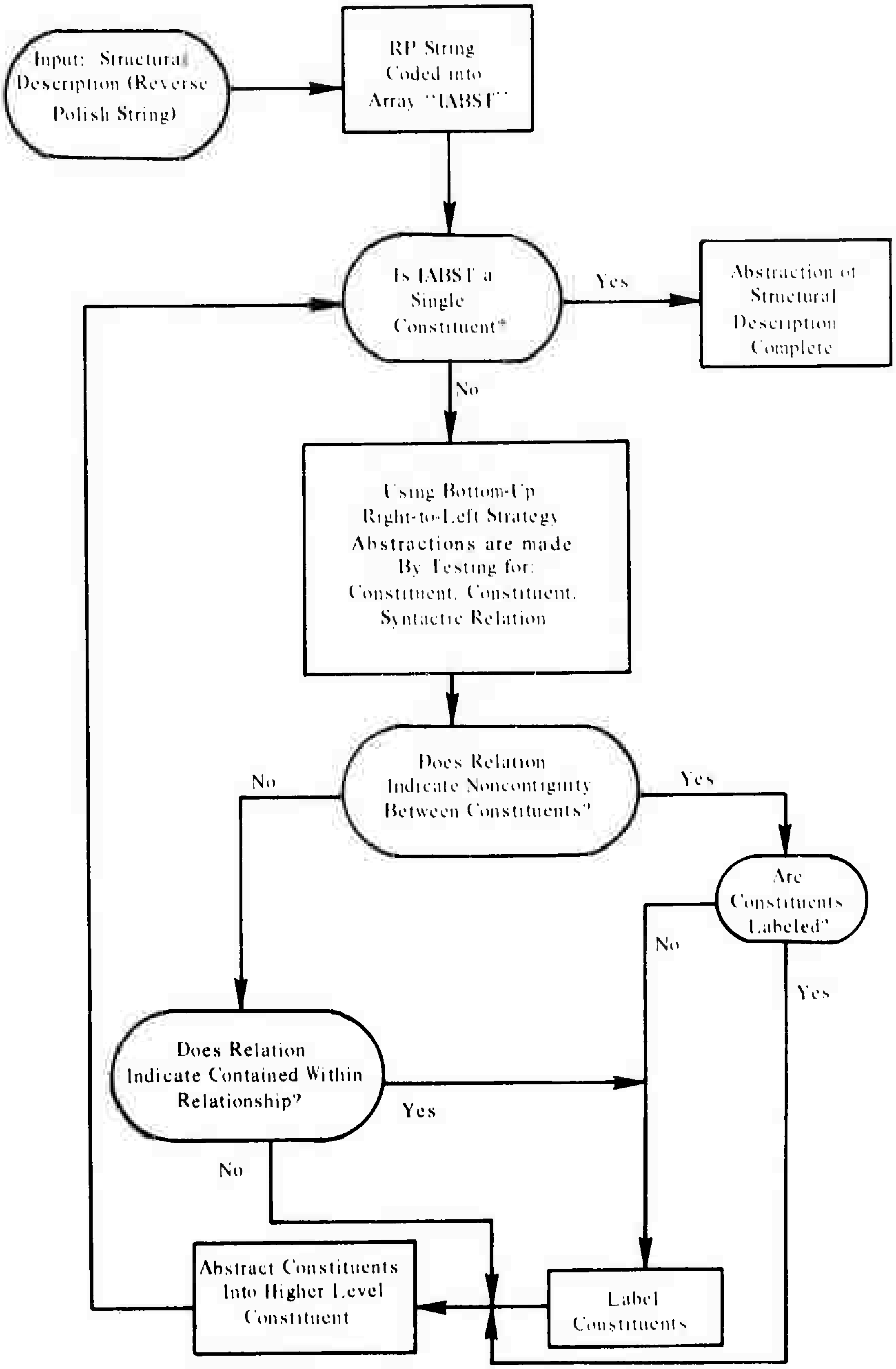

Input: Structural Description (Reverse Polish String)
RP String Coded into Array "IABST"
Is IABST a Single Constituent?
Yes
Abstraction of Structural Description Complete
No
Using Bottom-Up Right-to-Left Strategy Abstractions are made By Testing for: Constituent, Constituent, Syntactic Relation
Does Relation Indicate Noncontiguity Between Constituents?
No
Yes
Are Constituents Labeled?
No
Yes
Does Relation Indicate Contained Within Relationship?
Yes
No
Abstract Constituents Into Higher Level Constituent
Label Constituents

Figure 9.1.5 A Flow Diagram of the Program Which Segments and Abstracts a Figure

1. STORE	5. BULBOUS
2. HOUSE	6. OFFICE BLDG
3. SILO	7. UNICORN
4. TREE	

The structural descriptions can be of any complexity, having both *directional* recursion or *contained within* recursion. Figure 9.1.5 contains a flow diagram of the abstracting program. Section 9.1.2 is a listing of the program.

9.1.1 A Listing of the Program Which Obtains a Reverse Polish String from the Triplet Set

```
    DEFINE ((
                (REPOL(LAMBDA(MAIN)
                (PROG(RPS STATE ARG X REAM)
                (SETQ X (CAR MAIN))
    U  (COND(NULL ARG) (SETQ ARG(CONS(CAR X) NIL)))
                (T(SETQ ARG (CONS ( CAR X) (CDR)ARG)))))
    V  (SETQ ARG (CONS (CADR X) ARG))
       (COND((NULL RPS) (SETQ RPS ( CONS(CADDR X)NIL)))
                (T(SETQ RPS (CONS(CADDR X) RPS))))
       (COND((NULL STATE ) (SETQ STATE(CONS 1 NIL)))
                (T(SETQ STATE (CONS 1 STATE))))
       (SETQ MAIN(EFFACE X MAIN))
    A  (COND(( NULL MAIN) (GO E)) (T (SETQ REAM MAIN)))
    B  (SETQ X (CAR REAM))
       (COND((EQ(CAR ARG)(CAR X)) (GO V))
                ((EQ(CAR ARG) (CADR X)) (GO U))(T(SETQ REAM(CDR REAM))))
       (COND((NULL REAM) (GO E))(T(GO B)))
    E  (SETQ RPS (CONS(CAR ARG) RPS))
       (SETQ ARG (CDR ARG))
    F  (SETQ STATE (CONS(ADD1(CAR STATE))(CDR STATE)))
       (COND((EQUAL (CAR STATE)3)(SETQ STATE(CDR STATE)))
                (T(GO A)))
       (COND((NOT(NULL STATE))(GO F))(T(PRINT RPS))))))
       (EFFACE(LAMBDA(XX LL)
       (COND((NULL LL) NIL)((EQUAL XX (CAR LL))
                (CDR LL)) (T(RPLACD (LL(EFFACE XX (CDR LL)))))))))
```

9.1.2 A Listing of the Program Which Segments and Abstracts a Figure

```
      DIMENSION INPUT (400),NDATA (50,4)
      DIMENSION MINAB(10),ISS(1,3),IABST(200,3),MAXAB(10)
      DIMENSION IAB(3)
      DIMENSION HHOLD(3,3), ITITE (16)
C**********************************************************
C     COMPILE THE STRUCTURAL DESCRIPTION STRING TO
C     CODED TABULAR FORM
C**********************************************************
```

```
      DO 8000 IJKL = 1,10
      DO 41 J = 1,3
      DO 41 I = 1,200
   41 IABST(I,J) = 0
      READ (1,9049) ITITE
 9049 FORMAT (16A5)
      WRITE (2,9049) ITITE
      READ (1,9000) N
 9000 FORMAT (I1)
      NN = N*80
      READ(1,9001) (INPUT(I), I = 1,NN)
 9001 FORMAT(80A1)
      WRITE(2,9050)
 9050 FORMAT(25H0  STRUCTURAL DESCRIPTION )
      WRITE(2,9002) (INPUT(I), I = 1,NN)
 9002 FORMAT(1H 115A1)
      DO 5 I = 1,NN
    5 INPUT(I) = XABSF(INPUT(I))
C***********************************************************
C     STRING WILL BE PUT IN ARRAY NDATA WITH
C     CONSTITUENTS CODED
C     TESTING FOR THE TYPE CONSTITUENT WHERE ALPHANUMERIC
C     EQUIVALENTS ARE   79 = R, 83 = T, 63 = C, 65 = E, 26 = *
C***********************************************************
      J = 0
      K = 0
    9 J = J + 1
   10 IF(INPUT(J) - 7900000000) 20,100,20
   20 IF(HIPUT(J) - 8300000000) 30,200,30
   30 IF(INPUT(J ) - 6300000000) 40,300,40
   40 IF(INPUT(J) - 6500000000) 50,400,50
   50 IF(INPUT(J) - 2600000000) 8000,500,8000
C***********************************************************
C     RECTANGLE CODED AS 9
C***********************************************************
  100 K = K + 1
      L = 1
      NDATA(K,L) = 9
C***********************************************************
C     TESTING FOR DELIMITERS WHERE,
C     36 = (, 16 = ), 35 = ,, 90 = 0, 91 = 1,...
C***********************************************************
  101 J = J + 1
  102 IF(INPUT(J) - 3600000000) 103,101,103
  103 IF(INPUT(J) - 1600000000) 104,150,104
  104 IF(INPUT(J) - 3500000000) 105,101,105
  105 IF(INPUT(J) - 9000000000) 8000,106,106
 106  ISTAT = 0
C***********************************************************
C     CODING NUMBERS TO INTEGER FORM
C***********************************************************
 107  NUM = (INPUT(J)/100000000) – 90
      J = J + 1
      IF(INPUT(J) – 9000000000) 110,108,108
 108  NUM = NUM*10 + ((INPUT(J)/1000000000)– 90)
 109  J = J + 1
 110  L = L + 1
      NDATA(K,L) = NUM
      IF(ISTAT) 102,102,510
C***********************************************************
C     TESTING FOR TYPE OF TRIANGLE WHERE,
C     69 = T, 79 = R, 73 = L, 64 = D
C***********************************************************
 200  J = J + 1
      IF(INPUT)J) – 6900000000) 210,260,210
 210  IF(INPUT)J) – 7900000000) 220,235,220
 220  IF(INPUT(J) – 7300000000) 8000,248,8000
C***********************************************************
C     TR CODED AS 11
C     TRD CODED AS 12
C***********************************************************
 235  J = J + 1
      K = K + 1
      L = 1
      IF(INPUT(J) –6400000000) 237,236,237
 236  NDATA(K,L) = 12
      GO TO 101
 237  NDATA(K,L) = 11
      GO TO 102
C***********************************************************
C     TL CODED AS 13
C     TLD CODED AS 14
C***********************************************************
 248  J = J + 1
      K = k + 1
      L = 1
      IF(INPUT(J) – 6400000000) 250,249,250
 249  NDATA(K,L) = 14
      GO TO 101
 250  NDATA(K,L) = 13
      GO TO 102
```

```
C***********************************************************
C     ISOSCELES TRIANGLE CODED AS 10
C***********************************************************
 260  K = K + 1
      L = 1
      NDATA(K,L) = 10
      GO TO 101
C***********************************************************
C     CIRCLE CODED AS 7
C***********************************************************
 300  K = K + 1
      L =  1
      NDATA(K,L) = 7
      GO TO 101
C***********************************************************
C     ELLIPSE CODED AS 8
C***********************************************************
 400  K = K + 1
      L = 1
      NDATA(K,L) = 8
      GO TO 101
C***********************************************************
C     TESTING FOR SYNTACTIC RELATIONS
C     RELATIONS ARE CODED AS THEIR REPRESENTATIVE INTEGER
C***********************************************************
 150  J = J + 1
      IF(INPUT(J) - 3500000000) 800,9,800
 500  J = J + 1
      IF(INPUT(J) - 7900000000) 8000,505,8000
 505  J = J + 1
      IF(INPUT(J) - 9000000000) 8000,8000,506
 506  K = K + 1
      L = 1
      NDATA(K,L) = (INPUT(J)/100000000) - 90
      J = J + 1
      IF(INPUT(J) - 3600000000) 8000,507,8000
 507  J = J + 1
      IF(INPUT(J) - 9000000000) 8000,508,508
 508  ISTAT = 1
      GO TO 107
 510  IF(INPUT(J) - 1600000000) 511,150,511
 511  J = J + 1
      GO TO 510
C***********************************************************
C     CALCULATING AREA OF PRIMITIVES AND STORING THE
C     TABLE IN ARRAY IABST
C***********************************************************
```

```
 800  I = 1
      MINAB(I) = 1
      MAX = K
      DO 808 K = 1, MAX
      IABST(K,1) = NDATA(K,1)
      IABST(K,2) = NDATA(K,2)
      IF(NDATA(K,1) - 7) 808,803,804
 803  RAD = NDATA(K,3)/2
      IABST(K,3) = 3,14*RAD**2
      GO TO 808
 804  IF(NDATA(K,1) - 9) 803,805,807
 805  IABST(K,3) = NDATA(K,3)*NDATA(K,4)
      GO TO 808
 807  IABST(K,3) = NDATA(K,3)*NDATA(K,4)/2
 808  CONTINUE
C***********************************************************
C     TABULAR FORM OF STRING
C***********************************************************
      WRITE(2,9060)
9060  FORMAT(42H0    TABULAR FORM OF STRUCTURAL DESCRIPTION)
      WRITE(2,9061)
9061  FORMAT(47H          PRIMITIVE TYPE          NUMBER         A
     XREA)
      WRITE(2,666)((IABST(NI,NJ),NJ = 1,3),NI = 1,K)
 666  FORMAT(1H 3I15)
C***********************************************************
C     THE REMAINDER OF THE PROGRAM ABSTRACTS AND LABELS
C     THE FIGURE WHILE EACH FIGURE IS SEGMENTED FROM
C     THE ENTIRE SYNTACTIC STRING
C***********************************************************
      MAXAB(I) = K
C***********************************************************
C     TESTING FOR RELATION INDICATING NONCONTIGUITY
C***********************************************************
 813  IF(IABST(K,1) - 4) 809,809,810
 809  IF(IABST(K,2)) 821,821,812
 810  IF(IABST(K,1) - 6) 811,811,900
 812  ITIT = 1
 692  K = K-1
      IF(K-MINAB(I)) 8000,690,690
C***********************************************************
C     COMBINING TWO INDENTIFIED FIGURES WHERE THE
C     IDENTIFIED FIGURES ARE LABELED 20
C***********************************************************
 690  IF(IABST(K,I) - 20) 814,691,814
 691  ITIT = ITIT + 1
      IF(ITIT - 2) 692,692,693
 693  ISS(1,1) = 20
      GO TO 694
C**********************************************************
C     TESTING FOR  SSS,SSS,SYNTACTIC RELATION
C**********************************************************
 821  IT = 1
 815  K = K - 1
 818  IF(K - MINAB(I))  8000,816,816
 816  IF(IABST(K,1) - 6)  817,817,825
 817  IF(IABST(K,1) - 5)  814,811,811
 811  K = K - 1
      IF(IABST(K,1) - 6)  817,817,820
 820  IF(IABST(K,1) - 15)  823,900,823
 823  K = K - 1
      IF(IABST(K,1) - 15)  819,900,693
 819  IF(IABST(K,1) - 6)  817,817,900
 825  IF(IT - 1)  900,826,850
 826  IT = 2
      GO TO 815
C**********************************************************
C     ABSTRACTING TWO CONSTITUENTS TO A NEW CONSTITUENT
C     LABELED 15
C     ARRAY ISS CONTAINS NEW CONSTITUENT
C**********************************************************
 850  ISS(1,1) = 15
 694  IF(IABST(K,3) - IABST(K + 1,3)  855,855,857
 855  IF(IABST(K + 1,1) - 15)  852,851,851
 851  ISS(1,2) = IABST(K + 1,2)
      GO TO 861
 852  ISS(1,2) = IABST(K + 1,1)
      GO TO 861
 857  IF(IABST(K,1)- 15)  854,853,853
 853  ISS(1,2) = IABST(K,2)
      GO TO 861
 854  ISS(1,2) = IABST(K,1)
 861  IF(IABST(K + 2,1) - 5)  858,863,863
 858  ISS(1,3) = IABST(K,3) + IABST(K + 1,3)
      GO TO 864
 863  ISS(1,3) = IABST(K + 1,3)
C**********************************************************
C     ARRAY IHOLD CONTAINS  CONSTITUENTS WHICH WERE
C     USED IN THE MOST RECENT  ABSTRACTION
C**********************************************************
 864  IM = 0
      KOOK = K + 2
      DO 867  IIK = K, KOOK
      IM = IM + 1
      DO 867  JJ = 1,3
 867  IHOLD(IM,JJ) = IABST(IIK,JJ)
C*********************************************************
C     CONSTITUENTS USED IN THE ABSTRACTION ARE ZEROED OUT
C     AND REMOVED FROM THE NEW TABLE
C*********************************************************
      IABST(K,1) = 0
      IABST(K + 1,1) = 0
      DO 856 J = 1,3
 856  IABST(K + 2,J) = ISS(1,J)
7098  ISTAR = MINAB(I)
      IFIN = MAXAB(I)
      I = I + 1
      MINAB(I) = IFIN + 1
      NOM = IFIN
      DO 875 J = ISTAR,IFIN
      IF(IABST(J,1)) 8000,875,859
 859  NOM = NOM + 1
      DO 860 JJ = 1,3
 860  IABST(NOM,JJ) = IABST(J,JJ)
 875  CONTINUE
C*********************************************************
C     WRITE CUT OF RESULTS OF LAST ABSTRACTION
C*********************************************************
      MM = MINAB(I)
      IABNO = I - 1
      WRITE(2, 8011)  IABNO
8011  FORMAT(22H0  ABSTRACTION NUMBER,15)
      WRITE(2,880)((IABST(M,N),N = 1,3), M = MM, NOM)
 880  FORMAT (1H 3115)
      MAXAB(I) = NOM
      IT = 0
      GO TO 813
C*********************************************************
C     FIGURES ARE CLASSIFIED BY THEIR ABSTRACTED STRINGS
C     ACCORDING TO THE RELATIVE AREA AND RELATIONS
C     BETWEEN THE PRIMITIVES
C*********************************************************
 900  IF(IABST(K,1) - 20)  899,8000,899
 899  IF(IABST(K,1)-15)  927,905,927
 927  IF(IABST(K,1) - 9)  904,974,904
 904  IF(IABST(K,1) - 10)  8989,936,8989
 905  IF(IHOLD(1,1) - 15)  944,945,945
 944  IAB1 = IHOLD(1,1)
      GO TO 906
 945  IAB1 = IHOLD(1,2)
 906  IF(IHOLD(2,1) - 15)  947,948,948
 947  IAB2 = IHOLD(2,1)
      GO TO 970
 948  IAB2 = IHOLD(2,2)
 970  IF(IAB1-9)  901,915,901
 901  IF(IAB1 - 7)  902,920,902
 902  IF(IAB1 - 8)  903.920.903
 903  IF(IAB1 - 10)  8989.925,8989
 915  IF(IAB2 - 9)  8989,916,8989
 916  IF(IHOLD(3,1) - 5)  976,975,976
 974  IABST(K,2) = IABST(K,1)
 975  WRITE (2,977)
 977  FORMAT(16H FIGURE IS STORE
      GO TO 999
 976  WRITE(2,917)
 917  FORMAT(23H FIGURE IS OFFICE BLDG.)
      GO TO 999
 920  IF(IAB2 - 9)  8989,921,8989
 921  WRITE(2,922)
 922  FORMAT(18H FIGURE IS BULBOUS)
      GO TO 999
 925  IF(IAB2 -9)  8989,926,8989
 926  IF(IHOLD(3,1) - 1)  8989,930,8989
 930  IF(3*IHOLD(1,3) - IHOLD(2,3))  940,931,931
 931  IF(IHOLD(1,3) - 2*IHOLD(2,3))  933,936,936
 933  WRITE (2,935)
 935  FORMAT (16H FIGURE IS HOUSE)
      GO TO 999
 936  WRITE (2,937)
 937  FORMAT (15H FIGURE IS TREE)
      GO TO 999
 940  WRITE (2,941)
 941  FORMAT (12H FIGURE IS SILO TYPE)
      GO TO 999
8989  WRITE (2,8990)
8990  FORMAT (18H FIGURE IS UNICORN)
C************************************************************
C     A CLASSIFIED FIGURE IS CODED 20
C************************************************************
 999  IABST(K,I) = 20
      K = MAXAB(I)
      GO TO 813
8000  CONTINUE
      END
```

APPENDIX 9.2

A LISTING OF THE RULES OF THE SYNTACTIC AND SEMANTIC COMPONENTS

The following pages contain a listing of the rules which are used in SAP. The rules are presented in the order listed below.

1. The Syntactic Component Grammar K1*.
2. The Transformation Rules for Forming Artificial Lines.
3. The Transformation Rules for the Concatenation of Lines.
4. The Lexicon.
5. The Transformation Rules for the Triplet Set.
6. The Grammar K2* for L*.
7. The First Level of the Semantic Component Grammar K3*.
8. The Constraint Matrices for First Level of K3*.
9. The Second Level of the Semantic Component Grammar K3*.
10. The Constraint Matrices for the Second Level of K3*.

1. Syntactic Component K1* Grammar:

$\langle L_H(x,x)\rangle ::= \langle P_H(x)\rangle$

$\langle L_H(y,x)\rangle ::= \langle P_H(y)\rangle\langle L_H(z,x)\rangle \mid \langle L_H(y,z)\rangle\langle P_H(x)\rangle$

$\langle L_V(x,x)\rangle ::= \langle P_V(x)\rangle$

$\langle L_V(y,x)\rangle ::= \langle P_V(y)\rangle\langle L_V(z,x)\rangle \mid \langle L_V(y,z)\rangle\langle P_V(x)\rangle$

$\langle L_{DR}(x,x)\rangle ::= \langle P_{DR}(x)\rangle$

$\langle L_{DR}(y,x)\rangle ::= \langle P_{DR}(y)\rangle\langle L_{DR}(z,x)\rangle \mid \langle L_{DR}(y,z)\rangle\langle P_{DR}(x)\rangle$

$\langle L_{DL}(x,x)\rangle ::= \langle P_{DL}(x)\rangle$

$\langle L_{DL}(y,x)\rangle ::= \langle P_{DL}(y)\rangle\langle L_{DL}(z,x)\rangle \mid \langle L_{DL}(y,z)\rangle\langle P_{DL}(x)\rangle$

$\langle R(w,h,v)\rangle ::= \langle L_H(v_l,v_j)\rangle\langle L_V(v_i,v_k)\rangle\langle L_H(v_k,v_l)\rangle$
$\langle L_V(v_j,v_l)\rangle$ /RECTANGLE

$\langle TI(w,h,v)\rangle ::= \langle L_H(v_i,v_j)\rangle\langle L_{DR}(v_i,v_k)\rangle$
$\langle L_{DL}(v_k,v_j)\rangle$ / ISOSCELES TRIANGLE

$\langle TR(w,h,v)\rangle ::= \langle L_{DR}(v_l,v_k)\rangle\langle L_V(v_j,v_k)\rangle$
$\langle L_H(v_i,v_j)\rangle$ / RIGHT TRIANGLE

$\langle TL(w,h,v)\rangle ::= \langle L_{DL}(v_k,v_j)\rangle\langle L_H(v_l,v_j)\rangle$
$\langle L_V(v_i,v_k)\rangle$ / LEFT TRIANGLE

$\langle TRD(w,h,v)\rangle ::= \langle L_{DL}(v_i,v_k)\rangle\langle L_H(v_i,v_j)\rangle$
$\langle L_V(v_k,v_j)\rangle$ / RIGHT TRIANGLE DOWN

$\langle TLD(w,h,v)\rangle ::= \langle L_{DR}(v_k,v_j)\rangle\langle L_H(v_l,v_j)\rangle$
$\langle L_V(v_k,v_l)\rangle$ / LEFT TRIANGLE DOWN

$\langle C(w,h,v)\rangle ::= \langle L_H(v_l,v_j)\rangle\langle L_{DL}(v_j,v_k)\rangle\langle L_V(v_l,v_k)\rangle$
$\rangle L_{DR}(v_m,v_l)\rangle\langle L_H(v_n,v_m)\rangle\langle L_{DL}(v_p,v_n)\rangle$
$\langle L_V(v_p,v_r)\rangle\langle L_{DR}(v_r,v_i)\rangle$ / CIRCLE

$\langle E(w,h,v)\rangle ::= \langle L_H(v_l,v_j)\rangle\langle L_{DL}(v_j,v_k)\rangle\langle L_V(v_l,v_k)\rangle$
$\langle L_{DR}(v_m,v_l)\rangle\langle L_H(v_n,v_m)\rangle\langle L_{DL}(v_p,v_n)\rangle$
$\langle L_V(v_p,v_r)\rangle\langle L_{DR}(v_r,v_i)\rangle$ / ELLIPSE

2. Transformation Rules for Forming Artificial Lines:

$$L_H(v_i,v_j)\ ,\ L_V(v_j,v_k) \longrightarrow \left\{\begin{array}{l}(1)\ ,\ (2)\ ,\ L_H(v_j,v_m)\\ (1)\ ,\ (2)\ ,\ L_V(v_m,v_j)\end{array}\right\}$$

$$L^H(v_j,v_i)\ ,\ L_V(v_j,v_k) \longrightarrow \left\{\begin{array}{l}(1)\ ,\ (2)\ ,\ L_H(v_m,v_j)\\ (1)\ ,\ (2)\ ,\ L_V(v_m,v_j)\end{array}\right\}$$

$$L_H(v_i,v_j)\ ,\ L_V(v_k,v_j) \longrightarrow \left\{\begin{array}{l}(1)\ ,\ (2)\ ,\ L_H(v_j,v_m)\\ (1)\ ,\ (2)\ ,\ L_V(v_j,v_m)\end{array}\right\}$$

$$L_H(v_j,v_i)\ ,\ L_V(v_k,v_j) \longrightarrow \left\{\begin{array}{l}(1)\ ,\ (2)\ ,\ L_H(v_m,v_j)\\ (1)\ ,\ (2)\ ,\ L_V(v_j,v_m)\end{array}\right\}$$

$$L_V(v_i,v_j)\ ,\ L_{DL}(v_k,v_j) \longrightarrow (1)\ ,\ (2)\ .\ L_H(v_m,v_j)$$

$$L_V(v_i,v_j)\ ,\ L_{DR}(v_j,v_k) \longrightarrow (1)\ ,\ (2)\ ,\ L_H(v_j,v_m)$$

$$L_H(v_i,v_j)\ ,\ L_{DR}(v_j,v_k) \longrightarrow (1)\ ,\ (2)\ ,\ L_H(v_j,v_m)$$

$$L_H(v_i,v_j)\ ,\ L_{DL}(v_j,v_k) \longrightarrow (1)\ ,\ (2)\ ,\ L_V(v_m,v_j)$$

$$L_H(v_j,v_i)\ ,\ L_{DL}(v_k,v_j) \longrightarrow (1)\ ,\ (2)\ ,\ L_H(v_m,v_j)$$

$$L_H(v_j,v_i)\ ,\ L_{DR}(v_k,v_j) \longrightarrow (1)\ ,\ (2)\ ,\ L_V(v_m,v_j)$$

$$L_{DL}(v_j,v_i)\ ,\ L_V(v_j,v_k) \longrightarrow (1)\ ,\ (2)\ ,\ L_{DL}(v_m,v_j)$$

$$L_{DR}(v_i,v_j)\ ,\ L_V(v_j,v_k) \longrightarrow (1)\ ,\ (2)\ ,\ L_{DR}(v_j,v_m)$$

$$L_{DR}(v_j,v_i)\ ,\ L_V(v_j,v_k) \longrightarrow (1)\ ,\ (2)\ ,\ L_H(v_m,v_j)$$

$$L_{DL}(v_i,v_j)\ ,\ L_V(v_j,v_k) \longrightarrow (1)\ ,\ (2)\ ,\ L_H(v_j,v_m)$$

3. Transformation Rules for the Concatenation of Lines:

$$L_1(v_i,v_j),L_1(v_j,v_k),L_m(v_j,v_n) \longrightarrow L'_1(v_i,v_k),(3)$$

$$L_1(v_l,v_j),L_1(v_j,v_k),L_m(v_n,v_j) \longrightarrow L'_1(v_l,v_k),(3)$$

4. The Lexicon:

CIRCLE : $C(n_1,n_2n_3)$

Requirements : $XLENGTH(L_H(v_i,v_j))=XLENGTH(L_V(v_l,v_k))=$
$XLENGTH(L_H(v_n,v_m))=YLENGTH(L_V(v_p,v_r))$
$XLENGTH(L_{DL}(v_i,v_k))=XLENGTH(L_{DR}(v_m,v_l))=$
$XLENGTH(L_{DL}(v_p,v_n))=XLENGTH(LDR(v_r,v_j))=$
$YLENGTH(L_{DL}(v_j,v_k))=YLENGTH(L_{DR}(v_m,v_l))=$
$YLENGTH(L_{DL}(v_p,v_n))=YLENGTH(L_{DR}(v_r,v_j))$

Assignments : $n_1 = CCOUNT + 1$

$n_2 = XLENGTH(L_H(v_i,v_j))+2(XLENGTH(L_{DL}(v_i,v_k)))$

$n_3 = n_2$

and,

$BOT = L_H(v_n,v_m)$

$TOP = L_H(v_i,v_j)$

$LE = L_V(v_p,v_r)$

$RE = L_V(v_l,v_k)$

reference point = $\left(\dfrac{v_k - v_r}{2} + v_r,\ \dfrac{v_i - v_n}{2} + v_n\right)$

ELLIPSE : $E(n_1,n_2,n_3)$

Requirements : $XLENGTH(L_H(v_i,v_j))=(L_H(v_n,v_m))$
$YLENGTH(L_V(v_l,v_k))=YLENGTH(L_V(v_p,v_r))$
$XLENGTH(L_{DL}(v_j,v_k))=XLENGTH(L_{DR}(v_m,v_l))=$
$XLENGTH(L_{DL}(v_p,v_n))=XLENGTH(L_{DR}(v_r,v_i))$
$YLENGTH(L_{DL}(v_l,v_k))=YLENGTH(L_{DR}(v_m,v_l))=$
$YLENGTH(L_{DL}(v_p,v_n))=YLENGTH(L_{DR}(v_r,v_i))$

Assignments : $n_1 = ECOUNT + 1$
$n_2 = XLENGTH(L_H(v_i,v_j)) + 2(XLENGTH(L_{DL}(v_j,v_k)))$
$n_3 = n_2$
and,
$BOT = L_H(v_n,v_m)$
$TOP = L_H(v_i,v_j)$
$LE = L_V(v_p,v_r)$
$RE = L_V(v_l,v_k)$
reference point = $\frac{v_k - v_r}{2} + v_r, \frac{v_i - v_n}{2} + v_n$

RECTANGLE: $R(n_1,n_2,n_3)$

Requirements: $XLENGTH(L_H(v_i,v_j)) = XLENGTH(L_H(v_k,v_l))$
$YLENGTH(L_V(v_i,v_k)) = YLENGTH(L_V(v_j,v_l))$

Assignments : $n_1 = RCOUNT + 1$
$n_2 = XLENGTH(L_H(v_i,v_j))$
$n_3 = YLENGTH(L_V(v_i,v_k))$
and,
$BOT = L_H(v_i,v_j)$
$TOP = L_H(v_k,v_l)$
$LR = L_V(v_i,v_k)$
$RE = L_V(v_j,v_l)$
reference point = v_i

ISOSCELES TRIANGLE : $TI(n_1,n_2,n_3)$

Requirements : $XLENGTH(L_{DR}(v_i,v_k)) = XLENGTH(L_{DL}(v_k,v_j))$

Assignments: $n_1 = TICOUNT + 1$
$n_2 = XLENGTH(L_H(v_i,v_j))$
$n_3 = YLENGTH(L_{DR}(v_i,v_k))$
and,
$BOT = L_H(v_i,v_j)$
$TOP = v_k$
$LE = v_i$
$RE = v_j$
reference point = v_i

RIGHT TRIANGLE : $TR(n_1,n_2,n_3)$

Requirements : ϕ

$n_1 = TRCOUNT + 1$
$n_2 = XLENGTH(L_H(v_i,v_j))$
$n_3 = YLENGTH(L_V(v_j,v_k))$
and,
$BOT = L_H(v_i,v_j)$
$TOP = v_k$
$LE = v_i$
$RE = L_V(v_j,v_k)$
reference point $= v_i$

RIGHT TRIANGLE DOWN : $TRD(n_1,n_2,n_3)$

Requirements : ϕ

Assignments : $n_1 = TRDCOUNT + 1$
$n_2 = XLENGTH(L_H(v_i,v_j))$
$n_3 = YLENGTH(L_V(v_k,v_j))$
and,
$BOT = v_k$
$TOP = L_H(v_i,v_j)$
$LE = v_i$
$RE = L_V(v_k,v_j)$
reference point $= v_i$

LEFT TRIANGLE : $TL(n_1,n_2,n_3)$

Requirements : ϕ

Assignments : $n_1 = TLCOUNT + 1$
$n_2 = XLENGTH(L_H(v_i,v_j))$
$n_3 = YLENGTH(L_V(v_i,v_k))$
and,
$BOT = L_H(v_i,v_j)$
$TOP = v_k$
$LE = L_V(v_i,v_k)$
$RE = v_j$
reference point $= v_i$

LEFT TRIANGLE DOWN : $TLD(n_1,n_2,n_3)$

Requirements : ϕ

Assignments : $n_1 = TLDCOUNT + 1$
$n_2 = XLENGTH(L_H(v_i,v_j))$
$n_3 = YLENGTH(L_V(v_k,v_i))$
and,
$BOT = v_k$
$TOP = L_H(v_i,v_j)$
$LE = v_i$
$RE = L_V(v_k,v_j)$
reference point $= v_i$

ON TOP OF :	$X,Y,{*}R1(n_1,n_2 \frown n_3 \frown n_4)$
Requirement :	$X(BOT) \boxplus Y(TOP)$
Assignments :	$n_1 = 0$
	$n_2 = LE$
	if $XCOORD(X_{LE}) \geq XCOORD(Y_{LE})$
	then,
	$n_3 = XCOORD(X_{LE}) - XCOORD(Y_{LE})$
	$n_4 = RT$
	if $XCOORD(X_{LE}) > XCOORD(Y_{LE})$
	then,
	$n_3 = XCOORD(Y_{LE}) - XCOORD(X_{LE})$
	$n_4 = LT$
TO RIGHT OF :	$X,Y,{*}R3\ (n_1,n_2 \frown n_3 \frown n_4)$
Requirement :	$X(LE) \boxplus Y(RE)$
Assignments :	$n_1 = 0$
	$n_2 = BOT$
	of $YCOORD(X_{BOT}) \geq YCOORD(Y_{BOT})$
	then,
	$n_3 = YCOORD(Y_{BOT}) - YCOORD(X_{BOT})$
	$n_4 = DN$
	if $YCOORD(Y_{BOT}) < YCOORD(X_{BOT})$
	then,
	$n_3 = YCOORD(X_{BOT}) - YCOORD(Y_{BOT})$
	$n_4 = UP$
CONTAINED WITHIN :	$X,Y,{*}R5(n_1,n_2 \frown n_3 \frown n_4,n_5 \frown n_6 \frown n_7)$
Requirement :	$X(1_i) \boxplus Y(1_i)$
Assignments :	$n_1 = 0$
	$n_2 = LE$
	$n_3 = XCOORD(X_{LE}) - XCOORD(Y_{LE})$
	$n_4 = RT$
	$n_5 = BOT$
	$n_6 = YCOORD(X_{BOT}) - YCOORD(Y_{BOT})$
	$n_7 = UP$

5. Transformation Rules for Triplet Set:

Transformation Rules for Isolated Primitives

$$(\alpha,\beta,{*}R5),(\gamma,0,0) \longrightarrow (1),(\gamma,\beta,{*}R5)$$
$$(\alpha,\beta,{*}Ri),(\gamma,0,0) \longrightarrow (1),(\alpha,\gamma,{*}R5)$$
$$(\alpha,0,0),(\beta,0,0) \longrightarrow (\beta,\alpha,{*}R5)$$
$$(\alpha,\beta,{*}R5),(\gamma,\delta,{*}Ri) \longrightarrow (1),(2),(\gamma,\beta,{*}R5)$$

Transformation Rules for Missing Relations

$$(\alpha,\beta,{*}Ri),(\alpha,\gamma,{*}Ri) \longrightarrow (1),(2),(\beta,\gamma,{*}Rj)$$
$$(\alpha,\beta,{*}Ri),(\gamma,\beta,{*}Ri) \longrightarrow (1),(2),(\alpha,\gamma,{*}Rj)$$
$$\text{where } i \neq j.$$

Transformation Rules for Inconsistent Relations

$$(\alpha,\beta,{*}Ri),(\beta,\gamma,{*}R5),(\alpha,\gamma,{*}Ri) \longrightarrow (2),(3)$$
$$(\beta,\alpha,{*}Ri),(\beta,\gamma,{*}R5),(\gamma,\alpha,{*}Ri) \longrightarrow (2),(3)$$
$$(\alpha,\beta,{*}R5),(\alpha,\gamma,{*}Ri),(\beta,\delta,{*}Ri),(\gamma,\delta,{*}R5) \longrightarrow (1),(3),(4)$$
$$(\beta,\alpha,(R5),(\beta,\gamma,{*}R5),(\gamma,\alpha,{*}R5) \longrightarrow (2),(3)$$

6. The Grammar K2* for L*:

```
                       <name cir>::= C
                     <name ellip>::= E
                      <name rect>::= R
                      <name isos>::= TI
                    <name rt tri>::= TR
                <name rt tridown>::= TRD
                   <name lft tri>::= TL
               <name lft tridown>::= TLD
                          <zero>::= 0
                        <number>::= 1|2|3|4|5|6|7|8|9
                       <integer>::= <number>|<zero>|<number>|<integer>
                       <numtype>::= <integer>
          <horizontal dimension>::= <integer>
            <vertical dimension>::= <integer>
           <primitive argument >::= <numtype>,<horizontal dimension>,
                                         <vertical dimension>
                        <xcoord>::= <integer>
                        <ycoord>::= <integer>
                         <refpt>::= <xcoord,>,<ycoord>
 <reference primitive argument>::= <zero>,<horizontal dimension>,
                                      <vertical dimension>,<refpt>
                        <circle>::= <name cir>(<primitive argument>)|
                                        <name cir>(<reference primitive argument>)
                       <ellipse>::= <name ellip>(<primitive argument>)|
                                        <name ellip>(<reference primitive argument>)
                     <rectangle>::= <name rect>(<primitive argument>)|
                                         <name rect>(<reference primitive argument>)
           <isosceles triangle>::= <name isos>(<primitive argument>)|
                                         <name isos>(<reference primitive argument>)
               <right triangle>::= <name rt tri>(<primitive argument>)|
                                         <name rt tri>(<reference primitive argument>)
          <right triangle down>::= <name rt tridown>(<primitive argument>)|
                                         <name rt tridown>(<reference primitive argument>)
            <left triangle>::= <name lft tri>(<primitive argument>)|
                               <name lft tri>(<reference primitive argument>)
       <left triangle down>::= <name lft tridown>(<primitive argument>)|
                               <name lft tridown>(<reference primitive argument>)
                <primitive>::= <rectangle>|<isosceles triangle>|
                               <right triangle>|<right triangle down>|
                               <left triangle>|<left triangle down>|
                               <circle>|<ellipse>
                  <name sr>::= *R
                      <msr>::= <integer>
                     <vedg>::= TOP|BOT|HC
                     <hedg>::= LE|RE|VC
                      <msp>::= <integer>
                     <vdir>::= UP|DN
                     <hdir>::= LT|RT
                  <vrelpos>::= <vedg><msp><vdir>
                  <hrelpos>::= <hedg><msp><hdir>
                   <argvsr>::= <msr>,<hrelpos>
                   <arghsr>::= <msr>,<vrelpos>
      <vertical relation 1>::= <name sr>1(<argvsr>)
      <vertical relation 2>::= <name sr>2(<argvsr>)
    <horizontal relation 3>::= <name sr>3(<arghsr>)
    <horizontal relation 4>::= <name sr>4(<arghsr>)
        <vertical relation>::= <vertical relation 1>|
                               <vertical relation 2>
      <horizontal relation>::= <horizontal relation 1>|
                               <horizontal relation 2>
     <directional relation>::= <vertical relation>|
                               <horizontal relation>
                  <argcwsr>::= <zero>,<vrelpos>,<hrelpos>|
                               <zero>,<hrelpos>,<vrelpos>
<contained within relation 5>::= <name sr>5(<argcwsr>)
<contained within relation 6>::= <name sr>6(<argcwsr>)
  <contained within relation>::= <contained within relation 5>|
                               <contained within relation 6>
       <syntactic relation>::= <vertical relation>|
                               <horizontal relation>|
                               <contained within relation>
                     <hsym>::= *HSYM
                     <vsym>::= *VSYM
                     <symf>::= <hsym><hdir>|<vsym><vdir>
                       <rp>::= *RP
                      <mbr>::= <integer>
                      <nur>::= <integer>
                      <rpf>::= <rp><mbr><hdir><nur>|
                               <rp><mbr><vdir><nur>
```

```
                          <els>::= *ELS
                          <rds>::= *RDS
                        <elrdf>::= <els><num> | <rds><num>
                         <defs>::= *DEFS
                         <defi>::= *DEF<integer>
                        <defsf>::= <defi><...><defs>
                         <rote>::= *ROTE
                        <rotef>::= <rote><integer>
   <unary syntactic function>::= <symf> | <rpf> | <elrdf> | <rotef>
  <binary syntactic function>::= <defsf>
                          <sss>::= <primitive><primitive><syntactic relation> |
                                   <sss><primitive><syntactic relation> |
                                   <primitive><sss><syntactic relation> |
                                   <sss><sss><syntactic relation> |
                                   <sss><unary syntactic function> |
                                   DEFi <sss> DEFS
                           <ss>::= <sss>
```

7. The First Level of the Semantic Component Grammar K3*

```
                    <figure>::= <house type> | <silo type> |
                                <store type> | <tree type> |
                                <bulbous type> | <office type>
                <house type>::= <doghouse> | <shed> | <house> |
                                <church> | <garage> | <barn> |
                                <school> | <barbershop>
                 <silo type>::= <silo type> | <lighthouse> |
                                <bouy> | <tower>
                 <tree type>::= <tree> | <tent> | <radio> | <tower>
                <store type>::= <store>
               <office type>::= <office bldg> | <flag pole>
              <bulbous type>::= <water tower> | <barber pole> |
                                <tree> | <lamppost> | <sign>
                      <shed>::= <facade 1>/(1,1),(1,2),(2,2),
                                (2,3),(3,2),(3,3)
                    <garage>::= <facade 1>/(1,1),(1,2),(2,2),
                                (2,3),(3,2),(3,3)
                  <doghouse>::= <facade 1>/(1,1),(1,2),(2,2),
                                (2,3),(3,2),(3,3)
                     <house>::= <facade1> | <facade 2>/(1,1),(1,2),(2,2),
                                (2,3),(3,2),(3,3)
                    <school>::= <facade 1> | <facade 2>/(1,1),(1,2),(2,2),
                                (2,3),(3,2),(3,3)
                <barbershop>::= <facade 1> | <facade 2>/(1,1),(1,2),(2,2),
                                (2,3),(3,2),(3,3)
```

```
         <barn>::= <facade 1> | <facade 3>/(4,4),(4,10),(6,13),
                                           (7,7),(7,8),(8,7),
                                           (8,8),(9,6),(9,9),
                                           (10,9),(10,10),(13,13)
       <church>::= <cross><facade 4><vertical
                                   relation 1>/(1,1),(1,2),
                                     (1,3),(2,1),(2,2),(2,4),
                                     (3,2),(3,3),(4,4),(4,10),
                                     (5,4),(5,6),(6,5),(7,7),
                                     (7,8),(8,7),(8,8),(9,6),
                                     (9,9),(10,9),(10,10),
                                     (11,10),(11,12),(11,12),
                                     (12,10),(13,13)
         <silo>::= <facade 0>/(1,2),(1,3),(3,1),(3,3)
   <lighthouse>::= <facade 0> | <roof><facade 5><vertical
                                           relation 1>/(1,1),
                                           (1,2),(1,3),(2,1),
                                           (2,2),(2,3),(3,1),
                                           (3,2),(3,3)
        <tower>::= <facade 0> | <roof><facade 5><vertical
                                           relation 1>/(1,1),
                                           (1,2),(1,3),(2,1),
                                           (2,2),(2,3),(3,1),
                                           (3,2),(3,3)
         <buoy>::= <facade 0>/(1,1),(1,3),(3,1),(3,3)
         <tree>::= <facade 0> | <facade 6> | <facade 7>/(1,1),(1,2),(2,1),
                                                       (2,2),(2,3),(2,4),
                                                       (3,2),(4,2),(4,4)
         <tent>::= <isosceles triangle><isosceles triangle>
                      <contained within relation 5>
  <radio tower>::= <facade 0>/(1,1),(1,2)
        <store>::= <front 3>/(1,1),(2,1),(2,2)
  <office bldg>::= <front 5><front 3><vertical relation 1> |
                      <front 5><office bldg><vertical
                                              relation 1>/(5,6),(6,5)
    <flag pole>::= <flag><facade 6><horizontal
                                   relation 3>/(2,3),(2,4),
                                    (3,2),(3,3),(4,2)
  <water tower>::= <facade 6>/(1,2),(2,1),(2,2)
  <barber pole>::= <facade 6>/(1,2),(2,1),(2,2)
     <lamppost>::= <facade 6>/(1,2),(2,1),(2,2)
         <sign>::= <facade 6>/(1,2),(2,1),(2,2)
     <facade 0>::= <roof><panel><vertical relation 1>
     <facade 1>::= <roof><front 1><vertical relation 1>
     <facade 2>::= <roof><front 2><vertical relation 1>
     <facade 3>::= <loft><roof><contained within relation 5>
                      <front 3><vertical relation 1>
```

```
<facade 4>::= <steeple><front 4><vertical relation 1>
<facade 5>::= <panel><base><vertical relation 1>
<facade 6>::= <bulb 1><panel><vertical relation 1>
<facade 7>::= <bulb 2><panel><vertical relation 1>
<inside 1>::= <window> | <door> |
                 <window><door><horizontal relation 3> |
                 <inside 1><window><horizontal relation 3>
<inside 2>::= <double door> |
                <window><double door><vertical relation 1>
<inside 3>::= <window><window><directional relation> |
                 <window><inside 3><directional relation>
 <front 1>::= <panel> | <door><panel><contained within
                                        relation 5>
 <front 2>::= <front 1> | <inside 1><panel><contained within
                                              relation 5>
 <front 3>::= <front 2> | <inside 2><panel><contained within
                                              relation 5>
 <front 4>::= <front 3> | <right wing><front 3>
                 <horizontal relation 3><left wing>
                 <horizontal relation 3>
 <front 5>::= <inside 3><panel><contained within
                                    relation 5> | <panel>
<steeple>::= <roof> | <stained glass><roof><contained within
                                             relation 5>
<stained glass>::= <isosceles triangle>
          <roof>::= <isosceles triangle>
         <panel>::= <rectangle>
         <cross>::= <right arm><upright><horizontal relation 3>
                      <left arm><horizontal relation 3>
   <right arm>::= <rectangle>
     <upright>::= <rectangle>
    <left arm>::= <rectangle>
   <left wing>::= <right triangle><rectangle><vertical
                                              relation 1>
  <right wing>::= <left triangle><rectangle><vertical
                                              relation 1>
      <window>::= <rectangle>
 <double door>::= <left door><right door><horizontal relation 3>
       <doors>::= <rectangle>
   <left door>::= <rectangle>
  <right door>::= <rectangle>
        <flag>::= <rectangle>
      <bulb 1>::= <circle>
      <bulb 2>::= <ellipse>
        <base>::= <rectangle>
        <loft>::= <rectangle>
     <ellipse>::= E(n,h,v)
```

<circle>::= C(n,h,v)
<rectangle>::= R(n,h,v)
<isosceles triangle>::= TI(n,h,v)
<right triangle>::= TR(n,h,v)
<right triangle down>::= TRD(n,h,v)
<left triangle>::= TL(n,h,v)
<left triangle down>::= TLD(n,h,v)
<vertical relation 1>::= *R1(n_1,n_2)
<horizontal relation 3>::= *R3(n_1,n_2)
<contained within relation 5>::= *R5(n_1,n_2,n_3)

Doghouse (DH)
Shed (SH)
Garage (GG)

	roof	panel	door
roof	DH: } SD: } H(1)· W(1) GG: 2H(1) W(1)< 5H(1)	DH: } SD: } W(1)· W(2) GG: 4H(1) ·H(2) ·2H(1)	
panel		DH: } SD: } W(1)· H(1)· 1.5W(1) GG: H(1)· W(1)· 3H(1)	DH 2W(3) W(2) SD: H(3) H(2) GG: 1.5H(3) H(2)
door		DH: } SD: } GG: } *R5(0.BOT VC)	DH: H(3) W(3) SD: H(3) ·2.5W(3) GG: W(3) ·H(3)

House (HS)
School (SH)
Barbershop (BS)

	roof	panel	door
roof	HS: SH BS. } H(1) W(1)	HS· W(1)· W(2) SH: BS: } W(1) W(2)	
panel		HS: SH: BS: } .3W(2)· H(2)· 1.5W(2)	HS· SH: BS: } H(3)· H(2)
door		HS· BS: } *R5(0.BOT.n_3) SH: *R5(0.BOT.VC)	HS· SH· BS: } H(3) ·2W(3)

Church (CH)
Barn (BN)

	right arm	upright	left arm	roof	stained glass	window
right arm	CH· W(1) -2H(1)	CH. H(1) V(2)	$\text{CH}\begin{cases}\text{H(1) H(3)}\\ \text{W(1)- W(3)}\end{cases}$			
upright	$\text{CH:}\begin{cases}{}^{*}\text{R3}(0, n_2 \cap X \cap n_3)\\ .3\text{H}(2)\ \ X\ -4\text{H}(2)\end{cases}$	CH H(2) 5W(2)		CH 2H(2)- H(4)		
left arm						
roof				$\left.\begin{matrix}\text{CH·}\\ \text{BW:}\end{matrix}\right\}$ 3W(4) -H(4) W(4)		
stained glass				CH· *R5(0,BOT,VC)		CH: $\begin{matrix}\text{W(5)- W(6)}\\ \text{H(5) -H(6)}\end{matrix}$
window					$\text{CH·}\begin{cases}\exists(2)\ \text{iff}\ (3)\\ \exists(3)\ \text{iff}\ (2)\end{cases}$	
left door						
right door						
door (double door)						$\left.\begin{matrix}\text{CH.}\\ \text{BN·}\end{matrix}\right\}$ *R1(n_1,0,VC)
panel						
right wing						
left wing						
loft						

Church (CH)
Barn (BN)

	left door	right door	door (double door)	panel	right wing	left wing	loft
right arm							
upright							
left arm							
roof				CH BN } W(4) W(10)			
stained glass							
window							BN W(6) W(13)
left door	CH BN } H(6) 2W(6)	CH BN } H(7) H(8) W(7) W(8)					
right door	CH BN } R3(0,BOT)	CH BN } H(8) 2W(8)					
door (double door)			CH BN } H(9) W(9)				
panel			CH BN } R5(0 BOT VC)	CH BN } H(10) W(10)			
right wing				CH R3(0 BOT)		CH { H(11) H(12) W(11) W(12)	
left wing				CH R3(0 BOT)			
loft							BN H(4) W(4)

Silo (SO)
Lighthouse (LH)
Bouy (BY)
Tower (TW)

	roof	panel	base
roof	SO: LH BY TW: } H(1) W(1)	LH TW. } W(1) W(2)	SO: LH: BY: TW: } W(2) W(3)
panel	LH TW: } *R1(0.VC)	LH: TW: } H(2) W(2)	LH. TW. } W(2) W(3)
base	SO: LH BY. TW } *R1(0.VC)	SO: LH. BY: TW: } *R1(0.VC)	SO: LH: BY TW: } H(3) 3W(3)

Tree (TR)
Radio Tower (RT)

	roof	panel	bulb 1	bulb 2
roof	TR: $W(1) \leq H(1) < 5W(1)$ RT. $4W(1) \leq H(1) < 7W(1)$	TR. $3W(2) < W(1) < 5W(2)$ RT. $4H(2)\ \ H(1)\ \ 7H(2)$		
panel	$\left.\begin{matrix}\text{TR:}\\ \text{RT:}\end{matrix}\right\\}$ *R1(0.VC)	TR: $3W(2) < H(2) < W(2)$ RT: $W(2)\ \ H(2)$	TR: $6W(2) < W(3) < 3W(4)$	TR: $\left\\{\begin{matrix}4H(2) < H(4)\ \ H(2)\\ W(4) < W(2)\end{matrix}\right.$
bulb 1		TR. *R1(0.VC)		
bulb 2		TR: *R1(0.VC)		TR $W(4)\ \ H(4)\ \ 5W(4)$

Store (ST)
Office Bldg (OB)
Flag Pole (FP)

	door	panel	flag	bulb i	front 1	front 3
door	ST: OB: $\}$ 2W(1)≤H(1)≤4W(1)			FP: W(4) – W(2)		
panel	ST: *R5(0,BOT,n_3)	ST: H(2)· W(2)· 4H(2)	FP: 5H(3)· H(2)· 10H(3)			
flag		FP: *R3(0,TOP)	FP: H(3)· W(3)· 2H(3)			
bulb 1		FP: *R1(0,VC)				
front 5						OB: W(5) W(3)
front 3					OB: *R1(0,VC)	

Water Tower (WT)
Barber Pole (BP)
Lamp Post (LP)
Sign (SN)

	bulb 1	panel
bulb 1		WT: } 3W(2)- W(1)- 7W(2) SN: } 2H(1)- H(2)- 5H(1) BP: } W(2)- W(1)- 4W(2) LP: } 3H(1)- H(2)- 6H(1)
panel	WT: SN: BP: LP: } ***R**1(0,VC)	WT: SN: BP: LP: } 3W(2)- H(2)- 7W(2)

8. The Second Level of the Semantic Component Grammar K3*

```
               <scene>::= <city> | <county> | <home> | <backyard> | <farm> |
                           <school yard> | <church yard> |
                           <barbers> | <forest> | <field> | <camp> |
                           <shopping center>
                <city>::= <barbers><shopping center><horizontal relation> |
                          <home><school><horizontal relation> |
                          <school><church><horizontal relation>
             <country>::= <home><farm><horizontal relation>
                <home>::= <house><backyard><horizontal relation> |
                           <house><garage><horizontal relation> |
                           <house><doghouse><horizontal relation> |
                           <tree><home><horizontal relation> |
                           <house><lamppost><horizontal relation>/(1,1),
                                                 (1,2),(1,3),(1,4),(1,5),
                                                 (2,2),(2,4),(3,3),(3,4),
                                                 (3,6)
            <backyard>::= <garage><doghouse><horizontal relation> |
                          <doghouse><shed><horizontal relation> |
                          <tree><backyard><horizontal relation>/(2,2),
                                              (2,3),(2,4),(3,3),
                                              (3,4),(3,6)
                <farm>::= <farm><shed><horizontal relation> |
                          <farm><silo><horizontal relation> |
                          <farm><shed><horizontal relation> |
                          <tree><farm><horizontal relation>/(1,2),(1,3)
         <school yard>::= <school><flag pole><horizontal relation>
         <church yard>::= <church><tree><horizontal relation> |
                           <tree><church yard><horizontal relation>
             <barbers>::= <barbershop><barber pole><horizontal
                                                 relation>/(1,2)
              <forest>::= <tree><tree><horizontal relation> |
                           <tree><forest><horizontal relation>
               <field>::= <water tower><forest><horizontal relation> |
                           <tower><forest><horizontal relation> |
                         <radio tower><forest><horizontal
                                                relation>/(1,2),
                                                (1,3),(1,4),(2,3),
                                                (2,4)
                <camp>::= <tent><tent><horizontal relation> |
                        <tent><camp><horizontal relation>
     <shopping center>::= <store><sign><horizontal relation> |
                        <store><store><horizontal relation> |
                        <store><shopping center><horizontal relation> |
                        <shopping center><shopping center>
                        <horizontal relation>/(1,2)
```

Home (HM)
Backyard (BY)

	house	garage	doghouse	tree	lamp post	shed
house	HM: 3W(1)>H(1)	HM: H(2)=H(1)	HM: H(1)>3H(3)	HM: H(1)<H(4)	HM:H(1)≤H(4)≤1.5H(1)	
garage		HM: BY: } 2H(2) W(2)	BY: { W(2) 4W(3) H(2) 2H(3)	BY HM: } H(2) H(4)		
doghouse			HM: BY: } H(3) W(3)	HM: BY: } 2H(3) H(4) 5H(3)		BY· 2H(3) H(6)· 4H(3)
tree						
lamp post						
shed						

Farm (FM)

	barn	shed	silo
barn		H(2) H(1) 3H(2)	H(2) H(3) 2H(2)
shed			
silo			

Barbers(BB)

	barbershop	barber pole
barbershop		H(2) H(1) 2H(2)
barber pole		

Field(FD)

	water tower	tree	tower	radio tower
water tower		2H(2) H(1) 4H(2)	H(3) H(1) 1.5H(3)	H(1) H(4) 1.5H(1)
tree			H(2) H(3) 2H(2)	2H(2) H(3) 4H(2)
tower				
radio tower				

Shopping Center (SC)

	store	sign
store		1.5H(1) H(2) 3H(1)
sign		

10. REFERENCES

1. Clark, W.A., and C E Molnar, *A Description of the LINC*, Computers in Biomedical Research, Volume II, R.W Stacy and B Waxman (Eds.), Academic Press New York, N Y., 1965,35 64.
2. Wilkes, M A., and W A Clark, *Programming the LINC*, Computer Research Laboratory, Washington University, St. Louis, Missouri, June 1965.
3. McCarthy,J., P.W.Abrahams, D J Edwards, T P Hart, and M I Levin, *LISP 1.5 Programmer's Manual*, MIT Press, Cambridge, Massachusetts, 1962.
4. Weissman, C., *LISP 1.5 Primer*, Dickenson Publishing Company, Belmont, California, 1967.
5. Unger, S H., *Pattern Detection and Recognition*, Proceedings of the IRE, Vol. 47,No. 10, October 1959, *1737 1752*.
6. Chomsky, N., *Current Issues in Linguistic Theory* The Structure of Language, J. Fodor and J. Katz (Eds.), Prentice-Hall, New Jersey, 1964, *50 119*
7. Hockett, C.F., *The Problem of Universals in Language*. Universals of Language, J H Greenberg (Ed.), MIT Press, Cambridge, Massachusetts, 1963, *1 30*.
8. Chomsky, N., *Aspects of the Theory of Syntax*, MIT Press, Cambridge, Massachusetts, 1965.
9. Katz,J. and J. Fodor, *The Structure of a Semantic Theory*, Language, Vol. 39, *170 210*, April-June 1963. Reprinted in J.Fodor and J Katz (Eds.),The Structure of Language, Prentice-Hall, New Jersey, 1964.
10. Katz,J. and P. Postal, *An Integrated Theory of Linguistic Descriptions*, MIT Press, Cambridge, Massachusetts, 1964.
11. Bolinger, D., *The Atomization of Meaning*, Language, Vol. 41, No. 4, *555 573*, 1965.
12. Feder, Jerome, *The Linguistic Approach to Pattern Analysis A Literature Survey*, New York University, Bronx, New York, Report 400-130, February 1966.
13. Grimsdale, R.L., F.H.Summer, C.J.Tunis, T.Kilburn, *A System for the Automatic Recognition of Patterns*, Proceedings of the Institute of Electrical Engineers, Vol. 106, Part B, No. 26, *210 221*, March 1959, Reprinted in, Leonard Uhr (Ed.), Pattern Recognition, John Wiley and Sons, New York, 1966.
14. Kirsch, Russell A., *Computer Interpretation of English Text and Picture Patterns*, IEEE Transaction on Electronic Computers, Vol. EC-13, No. 4, *363 376*, August 1964.
15. Narasimhan, R., *A Linguistic Approach to Pattern Recognition*, Digital Computer Laboratory, University of Illinois, Urbana, Illinois, Report No. 121, July 10, 1962.
16. Narasimhan, R., *Syntactic Descriptions of Pictures and Gestalt Phenomena of Visual Perception*, Digital Computer Laboratory, University of Illinois, Urbana, Illinois, Report No. 142, July 25, 1963.
17. Narasimhan, R., J.R.Witsken, and H.Johnson, *Bubble Talk: The Structure of a Program for On-Line Conversation with ILLIAC III*, Digital Computer Laboratory, University of Ill. Urbana, Illinois, File No. 604, July 2, 1964.
18. Narasimhan, R., *Labeling Schemata and Syntactic Descriptions of Pictures*, Information and Control, Vol. 7, *151 179*, July 1964.
19. Narasimhan, R., *Syntax Directed Interpretation of Classes of Pictures*, Communications of the ACM, Vol. 9, No. 3, *166 173*, March 1966.
20. Freeman, Herbert, *On the Encoding of Arbitrary Geometric Configurations*, IRE Transactions on Electronic Computers, Vol. EC-10, No. 2, *260-268*, June 1961.

21. Freeman, Herbert. *Techniques for the Digital Computer Analysis of Chain-Encoded Arbitrary Plane Curves*, Proceedings of the National Electronics Conference, Vol. 17, *421-432*, October 1961.

22. Freeman, Herbert. *On the Digital Computer Classification of Geometric Line Patterns*, Proceedings of the National Electronics Conference, Vol. 18, *312-324*, October 1962.

23. Freeman, Herbert, and L. Garder, *Apictorial Jigsaw Puzzles: The Computer Solution of a Problem in Pattern Recognition*, IEEE Transactions on Electronic Computers, Vol. EC-13, No. 2, *118-127*, April 1964.

24. Eden, Murray, *On the Formalization of Handwriting*, Proceedings of Symposia in Applied Mathematics, American Mathematical Society, Vol. 12, *83-88*, 1961.

25. Eden, M., *Handwriting and Pattern Recognition*, IRE Transactions on Information Theory, IT-8, No. 2, *160-166*, 1962.

26. Breeding, Kenneth James, *Grammar for a Pattern Description Language*, Department of Computer Science, University of Illinois, Urbana, Illinois, Report No. 177, May 27, 1965.

27. Ledley, Robert S., *Thousand-Gate Computer Simulation of a Billion-Gate Computer*, Computer and Information Sciences, J.T.Tou and R.H.Wilcox (Eds.)' Spartan Books, Washington, D.C. *457-480*, 1964.

28. Ledley, Robert S., *High-Speed Automatic Analysis of Biomedical Pictures*, Science, Vol. 9, *216-223*, October 9, 1964.

29. Ledley, Robert S. and James B. Wilson, *Concept Analysis by Syntax Processing*, Proceedings of the American Document Institute Annual Meetings, Vol. 1, *1-8*, October 1964.

30. Ledley, Robert S. and Frank H. Ruddle, *Chromosome Analysis by Computer*, Scientific American, Vol. 214, No.4, *40-46*, April 1966.

31. Miller, W.F. and Alan C. Shaw, *A Picture Calculus*, Stanford University, Stanford, California, SLAC-PUB-358, October 1967.

32. George, J. E. and W.F.Miller, *String Descriptions of Data for Display*, Computer Science Department, Stanford University, Stanford, California, SLAC-PUB-383, January 1968.

33. Shaw, A.C., *The Formal Description and Parsing of Pictures*, Technical Report No. CS 94, Computer Science Department, Stanford University, Stanford, California, April 1968.

34. Martin, William A., *Syntax and Display of Mathematical Expressions*, Massachusetts Institute of Technology, Cambridge, Massachusetts, Memorandum MAC-M-257, July 29, 1965.

35. Anderson, Robert H., *Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics*, ACM Symposium on Interactive Systems for Experimental Applied Mathematics, Washington, D.C., August 26-28, 1967.

36. Anderson, Robert H., *Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics*, Ph.D. Thesis, Harvard University, Cambridge, Massachusetts, January 1968.

37. Clowes, M.B., *Perception, Picture Processing, and Computers*, Machine Intelligence 1, N.L. Collins and Donald Michie, (Eds.), American Elsevier Publishing Company, New York, 1967.

38. Clowes, M.B., *A Generative Picture Grammar*, Seminar Paper No. 6, Computing Research Section Commonwealth Scientific and Industrial Research Organization, Australia, April 1967.

39. Uhr, L.,*Pattern Recognition*, Electronic Information Handling, A.Kent and O.E.Taulbee, (Eds.), Spartan Books, Washington, D.C., *51-72*, 1965. Reprinted in L. Uhr (Ed.), Pattern Recognition, John Wiley and Sons, New York, 1966.

40. Lipkin, L., W.Watt, and R. Kirsch, *The Analysis, Synthesis, and Description of Biological Images*, Annals of the New York Academy of Sciences, Vol. 138, No. 3, *948-1012*, January 1966.

41. Busacher,Robert G., and Thomas L.Saaty, *Finite Graphs and Networks*, McGraw-Hill Company, New York, 1965.

42. Vernon, M.D., *A Further Study of Visual Perception*, Cambridge University Press, Cambridge, England, 1952.

43. Solomonoff, R., *A Formal Theory of Inductive Inference*, Information and Control, Vol. 7, *1-22, 224-254*, 1964.

44. Feldman. J., *First Thoughts on Grammatical Inference*, Stanford Artificial Intelligence Memo No. 55, Stanford University. Stanford, California, August 1967.

45. Standish, T.A., *A Data Definition Facility for Programming Languages*, Ph. D. Thesis. Carnegie Institute of Technology, Pittsburgh, Pennsylvania. May 1967.

46. Feigenbaum, E.A., *The Simulation of Verbal Learning Behavior*, Proceedings of WJCC. Vol. 19, *121 132*, 1961, Reprinted in. E.A.Feigenbaum and J Feldman, Computers and Thought, McGraw-Hill Book Company, New York, 1963.

47. Feigenbaum, E.A., and H.A. Simon, *Generalization of an Elementary Perceiving and Memorizing Machine*, Proceeding of IFIP Congress, 1962, North-Holland Publishing Co., Amsterdam, Holland, 1962. Available as RAND Corporation Paper P-2555, RAND Corporation, Santa Monica, California, March 1962.

48. Wynn, W.H., *An Information Processing Model of Certain Aspects of Paired-Associate Learning*, Ph. D. Thesis. University of Texas, Austin, Texas, January 1966.

49. Amon, A.H., *Decision Structures for Recognition*, Report No. 126, Digital Computer Laboratory, University of Illinois, Urbana, Illinois, October 1962.

50. Minsky, Marvin L., *Steps Toward Artificial Intelligence*, Proceedings of the IRE. Vol. 49. No. 1, *8-29*, January 1961. Reprinted in Computers and Thought, E.A.Feigenbaum, and J. Feldman (Eds.). McGraw-Hill, New York, 1963.

51. Feder, Jerome, *Linguistic Specification and Analysis of Classes of Patterns*, Department of Electrical Engineering. New York University, N.Y., Report 400-147, October, 1966.

52. Hankley, W.J. and J.T. Tou, "Automatic Fingerprint Interpretation and Classification Via Contextual Analysis and Topological Coding," *Pictorial Pattern Recognition*, G. Cheng, D. Pollock, and A. Rosenfeld (Eds.), Thompson Book Company, Washington D. C., pp.*411-456*, 1968.

53. Inselberg, Armond, *Syntax-Directed Pattern Analysis: A Literature Survey*, Technical Memorandum No. 46, Computer Systems Laboratory. Washington University, St. Louis. Missouri, August , 1968.

54. Kirsch, Russell A., (Chairman), *Discussion Summary on Graphical Languages*, Communications of the ACM, Vol. 9, No. 3, pp. *175-176*, March 1966.

55. Knoke, Peter J. and Richard G. Wiley, "A Linguistic Approach to Mechanical Pattern Recognition," *Digest of the First Annual IEEE Computer Conference*, Chicago, Illinois, *(142-144)*, September 6-8, 1967.

56. Ledley, R.S., F.H. Ruddle, J.B. Wilson, M. Belson. and J. Albarran, "The Case of the Touching and Overlapping Chromosomes," *Pictorial Pattern Recognition*, G. Cheng, R. Ledley. D. Pollock, and A. Rosenfeld, (Eds.), Thompson Book Company, Washington. D.C., *(87-97)*, 1968.